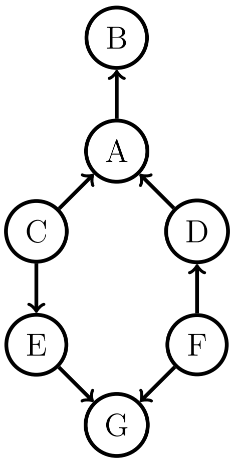
CS601 - Introduction to Artificial Intelligence
Assignment 1

Ian CHONG Wei Ming
ian.chong.2020@mitb.smu.edu.sg
+65-9680-8118

February 17, 2021

Question 1 Directed Bayesian Network

Given the following Bayesian network:



The truth of the following statements are as follows:

1a) $C \perp F \mid \{E, G\}$

TRUE. There exist two trails between C and F:

$$Trail1 : C \rightarrow \underline{\mathbf{E}} \rightarrow \underline{\mathbf{G}} \leftarrow F$$
$$Trail2 : C \rightarrow A \leftarrow D \leftarrow F$$

Trail 1 is *blocked*, since E and G are observed, and E blocks the trail from C to F, even if G is a collider and is *observed*. **Trail 2** is *blocked*, since E and G are observed, and A and its descendants (B) are not observed, and the two arrows from C and F lead into A, which is a *collider*.

Since both trails between C and F are *blocked*, C is *d-separated* from F, and C and F are therefore *conditionally independent* of each other.

1b) $C \perp F \mid \{D, G\}$

FALSE. There exist two paths between C and F:

$$Trail1 : C \rightarrow E \rightarrow \underline{\mathbf{G}} \leftarrow F$$
$$Trail2 : C \rightarrow A \leftarrow \underline{\mathbf{D}} \leftarrow F$$

Trail 1 is *active*, since D and G are observed, and C leads into G via E, and F also leads into G, giving rise to the *common effect*. **Path 2** is *blocked*, since D and G are observed, and A and its descendants (B) are not observed, and the two path arrows from C and F lead into A, which is a *collider*.

Since *not all* paths between C and F are *blocked*, C and F are therefore *dependent* on each other.

1c) $E \perp D \mid B$

FALSE. There are two trails between nodes E and D:

$$Trail1 : E \leftarrow C \rightarrow A \leftarrow D$$
$$Trail2 : E \rightarrow G \leftarrow F \rightarrow D$$

Trail 1 is *active* since B is observed and is a descendant of A which gives to the *common effect*.

Trail 2 is *blocked* since G exists on the trail which is a *collider*.

Since there is one *active* trail, E is not *d separated* from D and E is *dependent* on D.

1d) $E \perp D \mid \{\}$

TRUE. There are two trails between nodes E and D:

$$Trail1 : E \leftarrow C \rightarrow A \leftarrow D$$
$$Trail2 : E \rightarrow G \leftarrow F \rightarrow D$$

There are no observed nodes, and both trails are blocked by *colliders*. **Trail 1** is *blocked* by A, and **Trail 2** is blocked by G. Hence, E is *d-separated* from D, and they are therefore *independent* of each other.

1e) $A \perp F \mid \{C, D\}$

TRUE. There are two trails between nodes A and F:

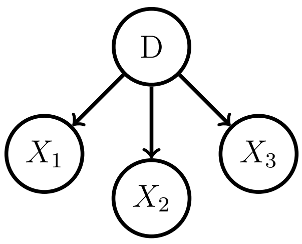
$$Trail1 : A \leftarrow \underline{\mathbf{D}} \rightarrow F$$
$$Trail2 : A \leftarrow \underline{\mathbf{C}} \rightarrow E \rightarrow G \leftarrow F$$

C and D are observed, and D lies on **Trail 1**, *blocking* the trail from F to A, while C lies on **Trail 2**, which *blocks* the trail from A to F. A and F are therefore *d-separated* and conditionally independent of one another.

Question 2 Box with three different dices

2a) Network Diagram and Random Variable Definitions

The Bayesian Network representing the dice and outcomes is as follows:



The variables X_1, X_2 and X_3 are all *independent* of one another since the expected outcome of a dice throw is dependent only on the characteristics of the dice selected. It does not matter how many times you throw the dice.

The interpretation of the random variables are as follows:

RV	**Domain**	**Interpretation**
D	{D_1,D_2,D_3}	Choice of dice between D_1, D_2 and D_3
X_1	{1,2,3,4}	The outcome of the first throw
X_2	{1,2,3,4}	The outcome of the second throw
X_3	{1,2,3,4}	The outcome of the third throw

2b) CPT by random variable

i) $P(D)$

D=1	D=2	D=3	**rowSum**
$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}$	**1**

ii) $P(X_1|D)$

	X_1=1	X_1=2	X_1=3	X_1=4	**colSum**
D=1	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	**1**
D=2	$\frac{1}{2}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	**1**
D=3	$\frac{1}{6}$	$\frac{1}{2}$	$\frac{1}{6}$	$\frac{1}{6}$	**1**

iii) $\mathbf{P}(X_2|D)$

	X_2=1	X_2=2	X_2=3	X_2=4	**colSum**
D=1	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	**1**
D=2	$\frac{1}{2}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	**1**
D=3	$\frac{1}{6}$	$\frac{1}{2}$	$\frac{1}{6}$	$\frac{1}{6}$	**1**

iv) $\mathbf{P}(X_3|D)$

	X_3=1	X_3=2	X_3=3	X_3=4	**colSum**
D=1	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	**1**
D=2	$\frac{1}{2}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	**1**
D=3	$\frac{1}{6}$	$\frac{1}{2}$	$\frac{1}{6}$	$\frac{1}{6}$	**1**

2c) $P(D|X_1 = 1, X_2 = 3, X_3 = 2)$

For a dice d, the probability that it gets picked given $X_1 = 1$, $X_2 = 3$ and $X_3 = 2$ is given by the following

$$\begin{aligned}P(D = d|X_1 = 1, X_2 = 3, X_3 = 2) &= \frac{P(D = d, X_1 = 1, X_2 = 3, X_3 = 2)}{P(X_1 = 1, X_2 = 3, X_3 = 2)} \\ &= \frac{P(D = d, X_1 = 1, X_2 = 3, X_3 = 2)}{\sum_{d\in D} P(D = d)P(X_1 = 1|D = d)P(X_2 = 3|D = d)P(X_3 = 2|D = d)}\end{aligned}$$

We can can calculate the denominator for the above as follows

$$\begin{aligned}P(X_1 = 1, X_2 = 3, X_3 = 2) &= \sum_{d\in D} P(D = d)P(X_1 = 1|D = d)P(X_2 = 3|D = d)P(X_3 = 2|D = d) \\ &= \left(\frac{1}{3}\cdot\frac{1}{4}\cdot\frac{1}{4}\cdot\frac{1}{4}\right) + \left(\frac{1}{3}\cdot\frac{1}{2}\cdot\frac{1}{6}\cdot\frac{1}{6}\right) + \left(\frac{1}{3}\cdot\frac{1}{6}\cdot\frac{1}{6}\cdot\frac{1}{2}\right) \\ &= 0.0052083 + 0.0046296 + 0.0046296 \\ &= 0.014468\end{aligned}$$

$$\begin{aligned}P(D = 1|X_1 = 1, X_2 = 3, X_3 = 2) &= \frac{0.0052083}{0.014468} \\ &= \underline{\mathbf{0.36}}\end{aligned}$$

$$\begin{aligned} P(D = 2|X_1 = 1, X_2 = 3, X_3 = 2) &= \frac{0.0046296}{0.014468} \\ &= \underline{\mathbf{0.32}} \end{aligned}$$

$$\begin{aligned} P(D = 3|X_1 = 1, X_2 = 3, X_3 = 2) &= \frac{0.0046296}{0.014468} \\ &= \underline{\mathbf{0.32}} \end{aligned}$$

Since the highest joint probability of the outcomes occurs when D=1, it *most likely* that D_1 was picked from the box.

Question 3 Constructing a BayesNet with pgmpy

The following random variables all have outcomes $\in \{0, 1\}$

Description	Random Variable
Yellow fingers	$Y \in \{0, 1\}$
Smoking	$S \in \{0, 1\}$
Cancer	$C \in \{0, 1\}$
Weakness	$W \in \{0, 1\}$
Radiation	$R \in \{0, 1\}$
Solar flare	$F \in \{0, 1\}$
Using microwave	$M \in \{0, 1\}$

3a) pgmpy code

The following code constructs the BayesNet

```
from pgmpy.models import BayesianModel
from pgmpy.factors.discrete import TabularCPD
# We first create a model which containts edges of the graph
model = BayesianModel([('S','Y'),
                       ('S','W'),
                       ('S','C'),
                       ('F','R'),
                       ('M','R'),
                       ('R','C')])

# Enter conditional probability distribution for each variable

# Prior probability for smoking P(S)
cpd_S = TabularCPD(variable='S', variable_card=2, values=[[0.85],[0.15]])

# Prior probability for solar flare P(F)
cpd_F = TabularCPD(variable='F', variable_card=2, values=[[0.99],[0.01]])

# Prior probability for micro wave P(M)
cpd_M = TabularCPD(variable='M', variable_card=2, values=[[0.05],[0.95]])

# Conditional probability for weakness W or P(W|S)
cpd_W = TabularCPD(variable='W',variable_card=2,
                   values = [[0.8,0.1],[0.2,0.9]],
                   evidence = ['S'],
                   evidence_card=[2])

# Conditional probability for radiation R or P(R|F,M)
cpd_R = TabularCPD(variable='R', variable_card=2,
                   values = [[0.9, 0.8, 0.8, 0.1],[0.1, 0.2, 0.2, 0.9]],
                   evidence = ['F','M'],
                   evidence_card=[2,2])

# Conditional probability for cancer C or P(C|S,R)
```

```
cpd_C = TabularCPD(variable='C', variable_card=2,
                    values = [[0.9, 0.4, 0.7, 0.1],[0.1, 0.6, 0.3, 0.9]],
                    evidence = ['S','R'],
                    evidence_card=[2,2])

# Conditional probability for yellow fingers Y or P(Y|S)
cpd_Y = TabularCPD(variable='Y',variable_card=2,
                    values = [[0.89,0.2],[0.11,0.8]],
                    evidence = ['S'],
                    evidence_card=[2])

# Add CPDs to the model
model.add_cpds(cpd_S, cpd_F, cpd_M, cpd_W, cpd_R, cpd_C, cpd_Y)

cpds = model.get_cpds()
for cpd in cpds:
    evidence = ",".join(cpd.variables[1:])
    if evidence:
        print(f"P({cpd.variables[0]}|{evidence})")
    else:
        print(f"P({cpd.variables[0]})")
    print(cpd)
```

Print of the results as follows

```
P(S)
+------+------+
| S(0) | 0.85 |
+------+------+
| S(1) | 0.15 |
+------+------+
P(F)
+------+------+
| F(0) | 0.99 |
+------+------+
| F(1) | 0.01 |
+------+------+
P(M)
+------+------+
| M(0) | 0.05 |
+------+------+
| M(1) | 0.95 |
+------+------+
P(W|S)
+------+------+------+
| S    | S(0) | S(1) |
+------+------+------+
| W(0) | 0.8  | 0.1  |
+------+------+------+
| W(1) | 0.2  | 0.9  |
+------+------+------+
P(R|F,M)
+------+------+------+------+------+
| F    | F(0) | F(0) | F(1) | F(1) |
+------+------+------+------+------+
| M    | M(0) | M(1) | M(0) | M(1) |
+------+------+------+------+------+
| R(0) | 0.9  | 0.8  | 0.8  | 0.1  |
+------+------+------+------+------+
| R(1) | 0.1  | 0.2  | 0.2  | 0.9  |
+------+------+------+------+------+
P(C|S,R)
+------+------+------+------+------+
| S    | S(0) | S(0) | S(1) | S(1) |
+------+------+------+------+------+
| R    | R(0) | R(1) | R(0) | R(1) |
+------+------+------+------+------+
| C(0) | 0.9  | 0.4  | 0.7  | 0.1  |
+------+------+------+------+------+
| C(1) | 0.1  | 0.6  | 0.3  | 0.9  |
+------+------+------+------+------+
P(Y|S)
+------+------+------+
| S    | S(0) | S(1) |
+------+------+------+
| Y(0) | 0.89 | 0.2  |
+------+------+------+
| Y(1) | 0.11 | 0.8  |
+------+------+------+
```

3b) Reponses

i) Draw the Bayesian Network

The random variables defined earlier are arranged in a Bayesian Network as follows:

$$\begin{tikzcd}
 & S \arrow[dl] \arrow[dd] \arrow[dr] & & F \arrow[dr] & & M \arrow[dl] \\
Y & & C & & R \arrow[ll] & \\
 & W & & & &
\end{tikzcd}$$

ii) Probability of cancer given weakness=1

Running the following code in `python`:

```
from pgmpy.inference import VariableElimination
infer = VariableElimination(model)
phi_query = infer.query(['C'], evidence={'W':1}, joint = False)
factor = phi_query['C']
print('Probability C|W')
print(factor)
```

Gives the following output:

```
Probability C|W
+------+----------+
| C    |   phi(C) |
+======+==========+
| C(0) |   0.7017 |
+------+----------+
| C(1) |   0.2983 |
+------+----------+
```

$P(C = 1|W = 1) = \underline{\mathbf{0.2983}}$

iii) Probability of smoking given cancer=1

Running the following:

```
phi_query = infer.query(['S'], evidence={'C':1}, joint = False)
factor = phi_query['S']
print('Probability S|C')
print(factor)
```

Gives the following output:

```
+------+----------+
| S    |   phi(S) |
+======+==========+
| S(0) |   0.7300 |
+------+----------+
| S(1) |   0.2700 |
+------+----------+
```

$P(S = 1|C = 1) = \underline{\mathbf{0.2700}}$

iv) Smoking and Radiation independent given cancer?

In order for S and R to be independent given C, we need to prove

$$P(S=1,R=1|C=1) = P(S=1|C=1) \times P(R=1|C=1)$$

To find $P(S-1,R=1|C=1)$, we run the following Python code:

```python
phi_query = infer.query(['S','R'],evidence={'C':1})
print(phi_query)
```

We achieve the following results:

```
+------+------+------------+
| S    | R    |   phi(S,R) |
+======+======+============+
| S(0) | R(0) |     0.2901 |
+------+------+------------+
| S(0) | R(1) |     0.4398 |
+------+------+------------+
| S(1) | R(0) |     0.1536 |
+------+------+------------+
| S(1) | R(1) |     0.1164 |
+------+------+------------+
```

We have $P(S=1,R=1|C=1) = \underline{\mathbf{0.1164}}$

Similarly, we achieve $P(S=1|C=1) = \mathbf{0.27}$ and $P(R=1|C=1) = \mathbf{0.5563}$. Multiplying the two probabilities we get

$$0.27 \times 0.5563 = 0.150201 \neq P(S=1,R=1|C=1)$$

Hence, S and R are not *conditionally independent*

Alternatively, there exists only one path between S and R:

$S \rightarrow C \leftarrow R$

Since C is observed, the path is *active* and S and R are *dependent* given C.

v) Probability of cancer without microwave

Running the following Python code:

```python
phi_query = infer.query(['C'],evidence={"M":0})
print(phi_query)
```

We get the output

```
+------+----------+
| C    |   phi(C) |
+======+==========+
| C(0) |   0.8180 |
+------+----------+
| C(1) |   0.1820 |
+------+----------+
```

$P(C = 1|M = 0) =$ **<u>0.1820</u>**

Question 4 Fashion MNIST dataset

Preamble

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras import layers
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.optimizers import Adagrad
from tensorflow.keras.losses import CategoricalCrossentropy
from tensorflow.keras.metrics import CategoricalAccuracy
from tensorflow.keras.callbacks import ReduceLROnPlateau, ModelCheckpoint

from sklearn.metrics import confusion_matrix, classification_report,
↪  accuracy_score
import numpy as np
from matplotlib import pyplot as plt
import seaborn as sns

directory = 'fashion_MNIST_checkpoint'
epochs = 30
optimizer =
↪  Adagrad(learning_rate=0.005,initial_accumulator_value=0.05,epsilon=1e-6)

model_checkpoint_callback = ModelCheckpoint(
    filepath=directory,
    save_weights_only=True,
    monitor='categorical_cross_entropy',
    mode='max',
    save_best_only=True)
reduce_lr = ReduceLROnPlateau(
    monitor='loss',
    factor = 0.2,
    patience = 5,
    min_lr = 0.0001
)
```

Download Fashion MNIST and Preprocessing

Fulfils 4a

```
fashion_mnist = tf.keras.datasets.fashion_mnist
(train_images, train_labels), (test_images, test_labels) =
↪  fashion_mnist.load_data()
train_images = train_images/255.0
test_images = test_images/255.0
print([train_images.shape,test_images.shape,train_labels.shape,test_labels.shape])
size = 28
train_count = len(train_images)
test_count = len(test_images)
input_shape = (size,size,1)
train_images_shaped = train_images.reshape(train_count,size,size,1)
test_images_shaped = test_images.reshape(test_count,size,size,1)
train_labels_shaped = to_categorical(train_labels)
```

```
test_labels_shaped = to_categorical(test_labels)
```

Neural Network Architecture

Fulfils 4b and 4c

```
model = Sequential()
model.add(layers.Dense(64,activation='relu'))
model.add(layers.Dropout(0.2))
model.add(layers.Dense(256, activation='relu'))
model.add(layers.Dropout(0.2))
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dropout(0.1))
model.add(layers.Flatten())
model.add(layers.Dense(10,activation='softmax'))
```

The activation function used in the last layer was `softmax` because this is a multilabel classification exercise

Compiling and Training the Network

Fulfils 4d

```
model.compile(
    optimizer=optimizer,
    loss="categorical_crossentropy",
    metrics=[CategoricalCrossentropy(),CategoricalAccuracy()])

history = model.fit(
    train_images_shaped,
    train_labels_shaped,
    epochs=epochs,
    callbacks=[model_checkpoint_callback,reduce_lr])
```

The loss function chosen was `categorical_crossentropy` because the labels were one hot encoded which changed the shape of the output.

Output as follows:

```
Epoch 1/30
1875/1875 [==============================] - 14s 7ms/step - loss: 0.7719 -
    categorical_crossentropy: 0.7719 - categorical_accuracy: 0.7425
WARNING:tensorflow:Can save best model only with categorical_cross_entropy
    available, skipping.
Epoch 2/30
1875/1875 [==============================] - 13s 7ms/step - loss: 0.4390 -
    categorical_crossentropy: 0.4390 - categorical_accuracy: 0.8456
WARNING:tensorflow:Can save best model only with categorical_cross_entropy
    available, skipping.
Epoch 3/30
1875/1875 [==============================] - 13s 7ms/step - loss: 0.4098 -
    categorical_crossentropy: 0.4098 - categorical_accuracy: 0.8554
```

```
WARNING:tensorflow:Can save best model only with categorical_cross_entropy
    available, skipping.
Epoch 4/30
1875/1875 [==============================] - 13s 7ms/step - loss: 0.3900 -
    categorical_crossentropy: 0.3900 - categorical_accuracy: 0.8612
WARNING:tensorflow:Can save best model only with categorical_cross_entropy
    available, skipping.
Epoch 5/30
1875/1875 [==============================] - 13s 7ms/step - loss: 0.3737 -
    categorical_crossentropy: 0.3737 - categorical_accuracy: 0.8660
WARNING:tensorflow:Can save best model only with categorical_cross_entropy
    available, skipping.
```

. . . Skipping for brevity's sake . . .

```
Epoch 26/30
1875/1875 [==============================] - 13s 7ms/step - loss: 0.3080 -
    categorical_crossentropy: 0.3080 - categorical_accuracy: 0.8887
WARNING:tensorflow:Can save best model only with categorical_cross_entropy
    available, skipping.
Epoch 27/30
1875/1875 [==============================] - 13s 7ms/step - loss: 0.2973 -
    categorical_crossentropy: 0.2973 - categorical_accuracy: 0.8910
WARNING:tensorflow:Can save best model only with categorical_cross_entropy
    available, skipping.
Epoch 28/30
1875/1875 [==============================] - 13s 7ms/step - loss: 0.2967 -
    categorical_crossentropy: 0.2967 - categorical_accuracy: 0.8908
WARNING:tensorflow:Can save best model only with categorical_cross_entropy
    available, skipping.
Epoch 29/30
1875/1875 [==============================] - 13s 7ms/step - loss: 0.2994 -
    categorical_crossentropy: 0.2994 - categorical_accuracy: 0.8914
WARNING:tensorflow:Can save best model only with categorical_cross_entropy
    available, skipping.
Epoch 30/30
1875/1875 [==============================] - 13s 7ms/step - loss: 0.2958 -
    categorical_crossentropy: 0.2958 - categorical_accuracy: 0.8927
WARNING:tensorflow:Can save best model only with categorical_cross_entropy
    available, skipping.
```

1) Confusion Matrix

The code to generate the confusion matrix in Figure 1 is as follows:

```python
test_labels_predicted = np.argmax(model.predict(test_images_shaped), axis=1)
cm = confusion_matrix(test_labels, test_labels_predicted)
plt.figure(dpi=100,figsize=(6,6))
ax=plt.gca()
sns.heatmap(cm, annot=True, fmt='.0f', cmap='Blues',ax=ax)
```

```
plt.title("Fashion MNIST Confusion Matrix")
plt.ylabel("Actual")
plt.xlabel("Predicted")
```

2) Loss Function

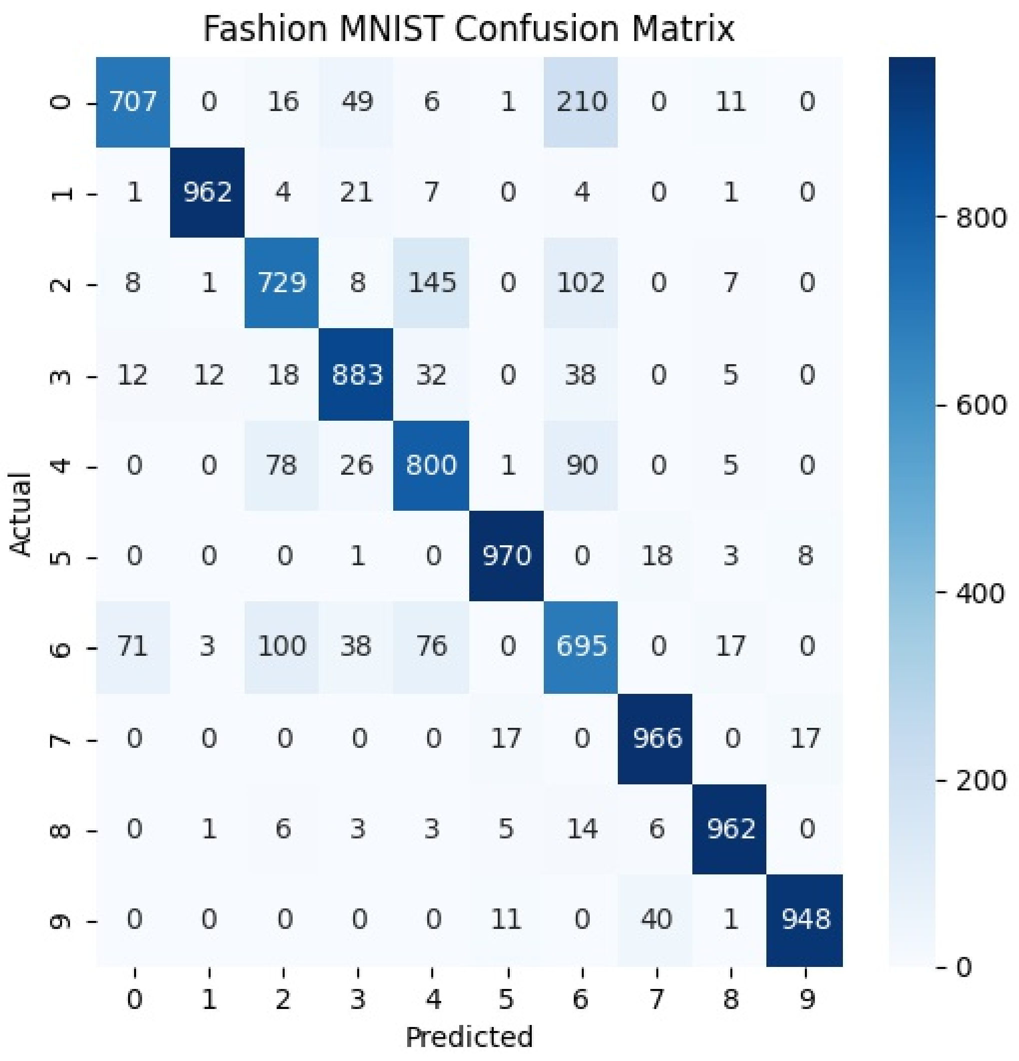
The loss function was `categorical_crossentropy`. Plot code to generate the result in Figure 2 is as follows:

```
import pandas as pd
training_data = pd.DataFrame(history.history)
plt.figure(dpi=100)
plt.title("Fashion MNIST Training Data")
plt.xlabel("Epoch")
ax = plt.gca()
training_data['categorical_crossentropy'].plot(ax=ax)
training_data['categorical_accuracy'].plot(ax=ax)
plt.legend()
```

3) Accuracy

Running the following line gives us accuracy on testing data of **86%**

```
model.evaluate(test_images_shaped,test_labels_shaped)
```



Actual \ Predicted	0	1	2	3	4	5	6	7	8	9
0	707	0	16	49	6	1	210	0	11	0
1	1	962	4	21	7	0	4	0	1	0
2	8	1	729	8	145	0	102	0	7	0
3	12	12	18	883	32	0	38	0	5	0
4	0	0	78	26	800	1	90	0	5	0
5	0	0	0	1	0	970	0	18	3	8
6	71	3	100	38	76	0	695	0	17	0
7	0	0	0	0	0	17	0	966	0	17
8	0	1	6	3	3	5	14	6	962	0
9	0	0	0	0	0	11	0	40	1	948

Figure 1: *Fashion MNIST* confusion matrix

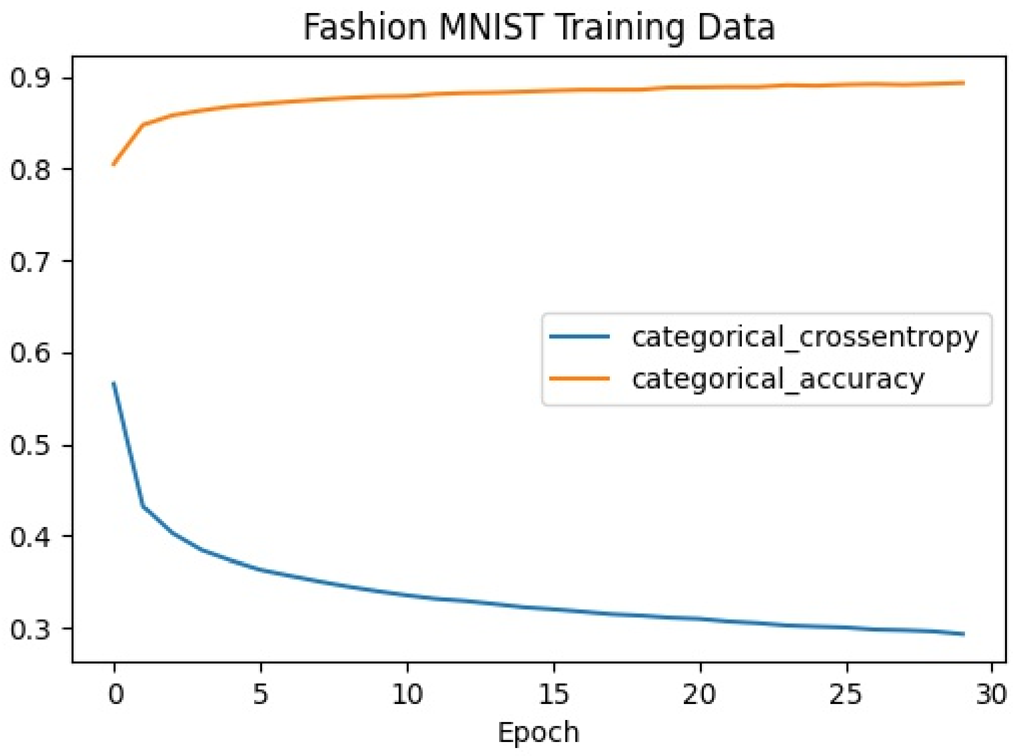


Figure 2: *Fashion MNIST* loss over time

Question 5 MobileNetV2 Transfer Learning

5a) python code

Preamble

```
import tensorflow as tf
from tensorflow.keras import Model
from tensorflow.keras.layers import Input, MaxPool2D, Dropout,Dense, Conv2D,
↪  Flatten
from tensorflow.keras.callbacks import ReduceLROnPlateau
from tensorflow.keras import Sequential
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import RMSprop
from tensorflow.keras.metrics import SparseCategoricalAccuracy,
↪  SparseCategoricalCrossentropy

from sklearn.metrics import accuracy_score, confusion_matrix
import seaborn as sns

import cv2

import sys
import numpy as np
import csv
import math
import os

from matplotlib import pyplot as plt
```

Key Notebook Parameters

I define key parameters of the notebook early on to facilitate transparency.

Input	Description
`use_sample`	Trigger sampling of data to facilitate faster prototyping
`sample_probability`	Chance that a row is selected to be a sample
`use_datagen`	Trigger `ImageDataGenerator` usage
`force_restart`	Ignore existing model and train new model from scratch
`epochs`	Number of times to train over the dataset
`metrics`	Metric classes to be included in the `.fit` phase
`final_layer_activation`	Activation function to use in the last layer
`optimizer`	Optimizer for model compilation
`loss`	Loss function for model compilation
`reduce_lr`	`LRReduceOnPlateau` callback to reduce learning rate after stagnation
`directory`	model saved directory

The classes for the dataset are the following:

Class No.	Description
0	airplane
1	automobile
2	bird
3	cat
4	deer
5	dog
6	frog
7	horse
8	ship
9	truck

The code is as follows:

```
use_sample = False
sample_probability = 0.05
use_datagen = False
force_restart = False

epochs = 10
metrics = [SparseCategoricalAccuracy("sparse_categorical_accuracy"),
↪  SparseCategoricalCrossentropy(name="sparse_categorical_cross_entropy")]

final_layer_activation = 'softmax'
optimizer = RMSprop(learning_rate=0.001)
loss = 'sparse_categorical_crossentropy'
reduce_lr = ReduceLROnPlateau(monitor='loss', factor=0.2,patience=3,
↪  min_lr=0.0005)

directory = "cs601_mobilenetv2_saved"

if not os.path.exists(directory):
    os.makedirs(directory)
    print("Directory created at [{}]!".format(directory))
else:
    "Directory [{}] exists!"
```

Import Dataset

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

# Load training data, labels; and testing data and their true labels
(train_images, train_labels), (test_images, test_labels) = cifar10.load_data()

if use_sample:
    mask = np.random.choice([False, True], len(train_images),p=[1 -
    ↪  sample_probability, sample_probability])
    train_images, train_labels = train_images[mask], train_labels[mask]

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

```
print ('Training data size:', train_images.shape,
        'Test data size', test_images.shape)
```

Visualize Dataset

```
%matplotlib inline
#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]])
```

Resize Images for `MobileNetV2` model

```
# Upsize images to 96x96 for use with MobileNetV2
minSize = 96 #minimum size requried for mobileNetV2
class_num = len(class_names)
image_num_train = len(train_images)
image_num_test = len(test_images)

# You may use cv2 package. Look for function:
resized_train_images = np.zeros((image_num_train, minSize, minSize, 3),
                                    dtype=np.float32)

resized_test_images = np.zeros((image_num_test, minSize, minSize, 3),
                                    dtype=np.float32)

for index, image in enumerate(train_images):
    resized_train_images[index]=cv2.resize(image,dsize=(minSize, minSize),
    ↪   interpolation=cv2.INTER_AREA)

for index, image in enumerate(test_images):
    cv2.resize(i, dsize=(minSize, minSize),interpolation=cv2.INTER_AREA)
    resized_test_images[index]=cv2.resize(image,dsize=(minSize, minSize),
    ↪   interpolation=cv2.INTER_AREA)
```

Download `MobileNetV2` model

The code here fulfils part 5a and 5b and 5c.

```
def make_model():
    """Create new MobileNetV2 model with frozen layers
    """
    # Initiate MobileNetV2 and freeze layers
    base_model = MobileNetV2(weights='imagenet', include_top=False,
    ↪   input_shape=input_shape)
    base_model.trainable = False

    model = Sequential()
    model.add(base_model)
    model.add(Conv2D(filters=16, kernel_size=(5, 5),strides=(2,2), padding='same',
    ↪  activation='relu'))
    model.add(Conv2D(filters=16, kernel_size=(5, 5),strides=(2,2), padding='same',
    ↪  activation='relu'))
    model.add(MaxPool2D(pool_size=(4, 4),strides=(2,2), padding='same'))
    model.add(Dropout(0.1))
    model.add(Conv2D(filters=32, kernel_size=(4, 4), padding='same',
    ↪  activation='relu'))
    model.add(Conv2D(filters=32, kernel_size=(4, 4), padding='same',
    ↪  activation='relu'))
    model.add(MaxPool2D(pool_size=(3, 3), padding='same'))
    model.add(Dropout(0.1))
    model.add(Conv2D(filters=64, kernel_size=(3, 3), padding='same',
    ↪  activation='relu'))
    model.add(Conv2D(filters=64, kernel_size=(3, 3), padding='same',
    ↪  activation='relu'))
    model.add(MaxPool2D(pool_size=(2, 2), padding='same'))
    model.add(Dropout(0.1))
    model.add(Dense(128, activation='relu'))
    model.add(Flatten())
    model.add(Dense(class_num, activation=final_layer_activation))
    return model

# Train new model if force_restart is True
# Else, if existing model detected, load it for further training
# Finally, create new model if all else fails

if force_restart:
    print("Force restart enabled")
    model = make_model()
    print("Created new model!")
else:
    try:
        print("Loading existing model from [{}]".format(directory))
        model = tf.keras.models.load_model(directory)
        print("Loaded model successfully!")
    except:
        model = make_model()
        print("Created new model!")
```

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

Fulfils part 5d

```
model.compile(optimizer=optimizer,loss=loss,metrics=metrics)
model_checkpoint_callback = tf.keras.callbacks.ModelCheckpoint(
    filepath=os.path.join(directory,"checkpoints"),monitor="val_loss")
history = model.fit(
    x=resized_train_images,
    y=train_labels,
    epochs=epochs,
    callbacks=[reduce_lr,model_checkpoint_callback])
```

`model.summary()` returns the following:

```
model.compile(optimizer=optimizer,loss=loss,metrics=metrics)
model.summary()
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
mobilenetv2_1.00_96 (Functio (None, 3, 3, 1280)        2257984   
_________________________________________________________________
conv2d (Conv2D)              (None, 2, 2, 16)          512016    
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 1, 1, 16)          6416      
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 1, 1, 16)          0         
_________________________________________________________________
dropout (Dropout)            (None, 1, 1, 16)          0         
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 1, 1, 32)          8224      
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 1, 1, 32)          16416     
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 1, 1, 32)          0         
_________________________________________________________________
dropout_1 (Dropout)          (None, 1, 1, 32)          0         
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 1, 1, 64)          18496     
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 1, 1, 64)          36928     
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 1, 1, 64)          0         
_________________________________________________________________
dropout_2 (Dropout)          (None, 1, 1, 64)          0         
_________________________________________________________________
dense (Dense)                (None, 1, 1, 128)         8320      
_________________________________________________________________
flatten (Flatten)            (None, 128)               0         
_________________________________________________________________
dense_1 (Dense)              (None, 10)                1290      
=================================================================
Total params: 2,866,090
Trainable params: 608,106
Non-trainable params: 2,257,984
_________________________________________________________________
```

Training Output

```
Epoch 1/50
1563/1563 [==============================] - 122s 76ms/step - loss: 1.4577
    - sparse_categorical_accuracy: 0.4273 -
    sparse_categorical_cross_entropy: 1.4577
```

```
INFO:tensorflow:Assets written to: cs601_mobilenetv2_saved/checkpoints/
   assets
Epoch 2/50
1563/1563 [==============================] - 115s 73ms/step - loss: 0.8207
   - sparse_categorical_accuracy: 0.7390 -
   sparse_categorical_cross_entropy: 0.8207
INFO:tensorflow:Assets written to: cs601_mobilenetv2_saved/checkpoints/
   assets
Epoch 3/50
1563/1563 [==============================] - 117s 75ms/step - loss: 0.6683
   - sparse_categorical_accuracy: 0.7889 -
   sparse_categorical_cross_entropy: 0.6683
INFO:tensorflow:Assets written to: cs601_mobilenetv2_saved/checkpoints/
   assets
Epoch 4/50
1563/1563 [==============================] - 106s 68ms/step - loss: 0.5803
   - sparse_categorical_accuracy: 0.8214 -
   sparse_categorical_cross_entropy: 0.5803
INFO:tensorflow:Assets written to: cs601_mobilenetv2_saved/checkpoints/
   assets
Epoch 5/50
1563/1563 [==============================] - 124s 79ms/step - loss: 0.5209
   - sparse_categorical_accuracy: 0.8424 -
   sparse_categorical_cross_entropy: 0.5209
INFO:tensorflow:Assets written to: cs601_mobilenetv2_saved/checkpoints/
   assets
```

. . . Leaving out the middle for brevity's sake . . .

```
Epoch 46/50
1563/1563 [==============================] - 99s 64ms/step - loss: 0.2797 -
    sparse_categorical_accuracy: 0.9544 - sparse_categorical_cross_entropy
   : 0.2797
INFO:tensorflow:Assets written to: cs601_mobilenetv2_saved/checkpoints/
   assets
Epoch 47/50
1563/1563 [==============================] - 94s 60ms/step - loss: 0.2568 -
    sparse_categorical_accuracy: 0.9564 - sparse_categorical_cross_entropy
   : 0.2568
INFO:tensorflow:Assets written to: cs601_mobilenetv2_saved/checkpoints/
   assets
Epoch 48/50
1563/1563 [==============================] - 96s 61ms/step - loss: 0.2764 -
    sparse_categorical_accuracy: 0.9557 - sparse_categorical_cross_entropy
   : 0.2764
INFO:tensorflow:Assets written to: cs601_mobilenetv2_saved/checkpoints/
   assets
Epoch 49/50
1563/1563 [==============================] - 94s 60ms/step - loss: 0.2971 -
    sparse_categorical_accuracy: 0.9542 - sparse_categorical_cross_entropy
   : 0.2971
INFO:tensorflow:Assets written to: cs601_mobilenetv2_saved/checkpoints/
   assets
Epoch 50/50
1563/1563 [==============================] - 96s 61ms/step - loss: 0.3920 -
    sparse_categorical_accuracy: 0.9555 - sparse_categorical_cross_entropy
   : 0.3920
INFO:tensorflow:Assets written to: cs601_mobilenetv2_saved/checkpoints/
   assets
```

5b) Explanation

i) Write how you extended the MobileNetV2 model

- **How many layers you added:** I added 14 layers as the images were larger than the `Fashion MNIST` data set, and had more features to extract.
- **What type of layers:** I used `Conv2D` and `MaxPooling2D` layers to extract features and `Dropout` layers to ensure the weights of neurons do not go to zero as well as to reduce overfitting.
- **How many nodes per layer:** I used **128** nodes at the last layer.
- **Their activation function:** I used `relu` for all the hidden layers and `softmax` for the output layer.

ii) Plot the loss function value with respect to the epoch number on the training data. How did you decide when to terminate training?

The model was trained over 50 epochs on a Colab Pro instance.

The loss stopped improving past the **23rd epoch** as per Figure 3.

iii) Show the accuracy of the trained classifier over the entire testing dataset.

Training accuracy was high at over **95%** after **28 epochs** of training, but this did not improve further as shown in Figure 4.

This happened after the *learning rate* was dropped following a plateau on the **23rd epoch**, triggering the `ReduceLROnPlateau` callback. (see Figure 5)

The lower learning rate helped the algorithm to get past the *local minima* and reach a better outcome.

Running the following code gave the accuracy on the test data set.

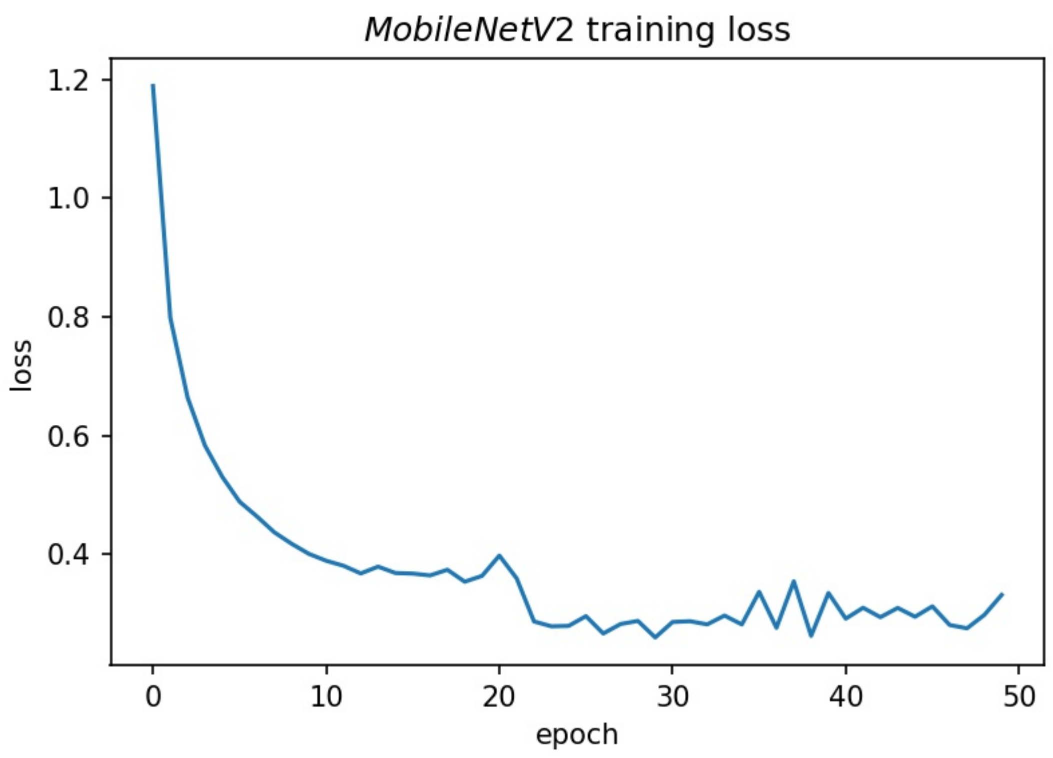


Figure 3: `MobileNetV2` training loss

Actual	Predicted	Count
Class 3 - `cat`	Class 5 - `dog`	163
Class 5 - `dog`	Class 3 - `cat`	160
Class 6 - `frog`	Class 2 - `bird`	99
Class 4 - `deer`	Class 2 - `bird`	94
Class 6 - `frog`	Class 3 - `cat`	89

Table 1: Common mistakes made by trained model

```
results = model.evaluate(resized_test_images, test_labels, batch_size=128)
```

The score deteriorated to 78% on the training set. This suggests *overfitting* or inappropriate biases being introduced from the original `imagenet` data that the pre-model was trained on.

According the confusion matrix (Figure 6), the the most common mistake made by the model was mistaking a cat for a dog. See Table 1 for details.

5c) Addendum: Further observations

While earlier results were not documented, the following oberservations were made when not freezing the `MobileNetV2` base model layer and training instead all layers from start to finish:

1. The training time was an order of magnitude larger, which was logical given that the number of layers in `MobileNetV2` were not trivial. Freezing the base model would have required less updating of the weights.
2. The *training accuracy* was lower at 85%. However, when evaluted, the *testing accuracy* was also similar between 80 – 85%. As observed above, freezing the *MobileNetV2* layers proved to increase *training accuracy*, but the *testing accuracy* dropped significantly, suggesting some overfitting has occurred.
3. The most common misclassification instances were also of a different nature. The earlier models which did not retain the original "imagenet" weights mistook automobiles *(class 1)* for trucks *(class 9)* more. This suggests the base model was trained on more varied images relating to vehicles. With Google's reputation for developing self driving cars, it is plausible that the model had exposure to better data on the road.

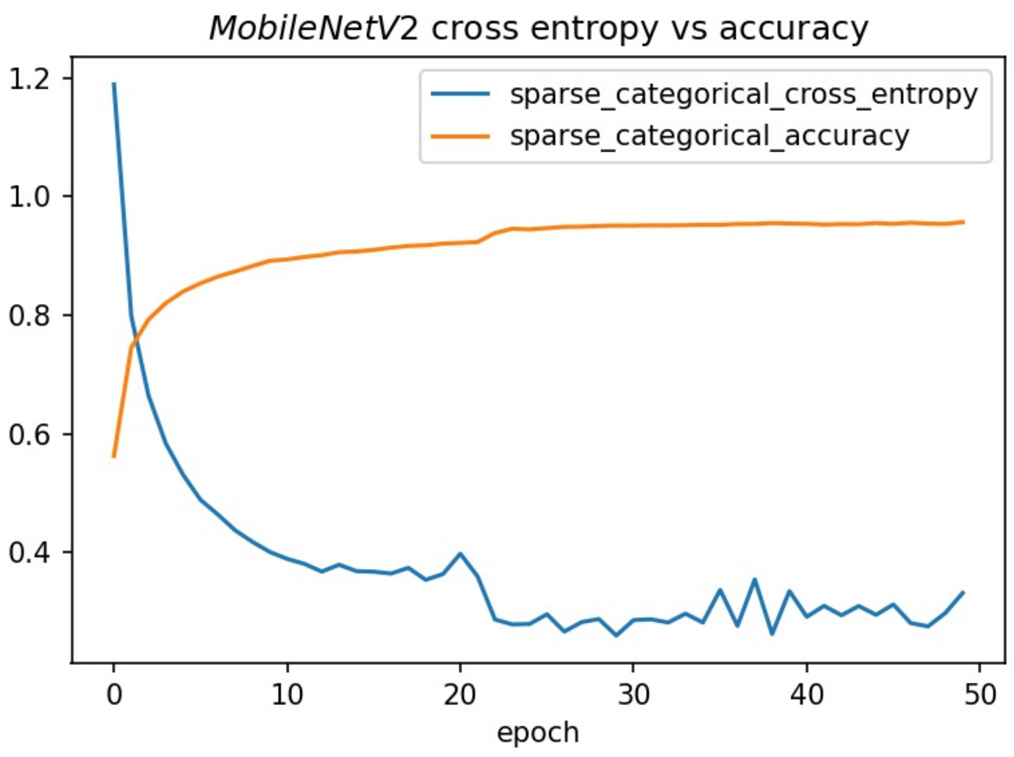


Figure 4: `MobileNetV2` training cross entropy vs accuracy

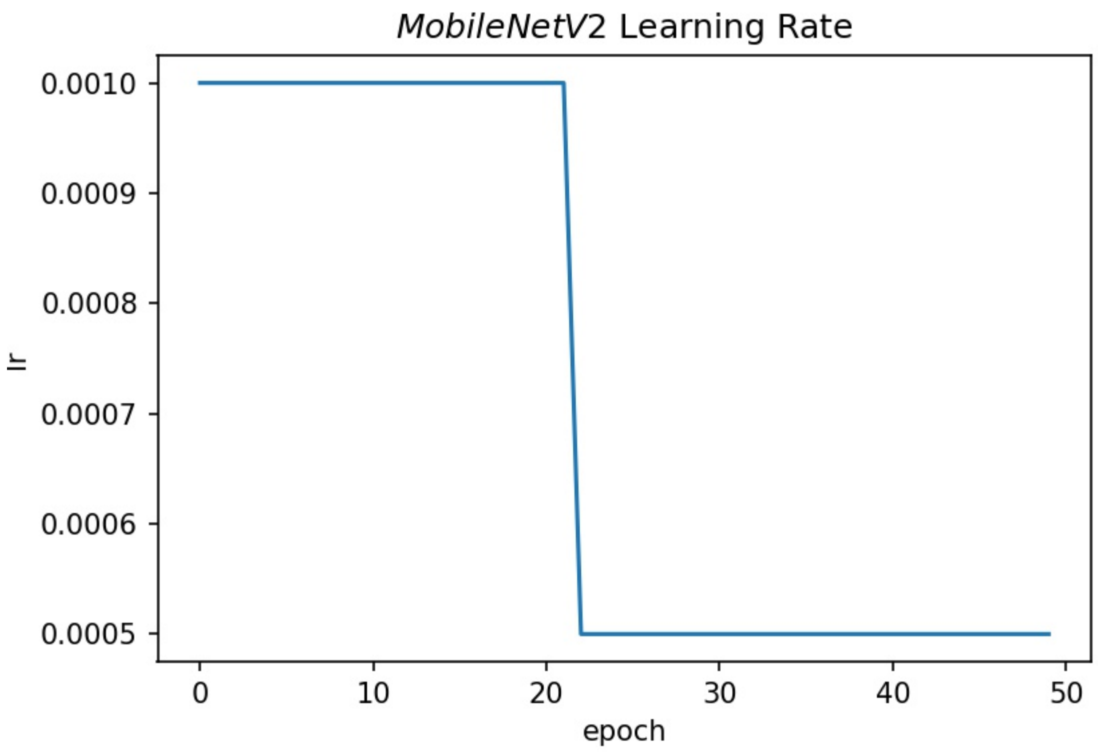


Figure 5: `MobileNetV2` training learning rate

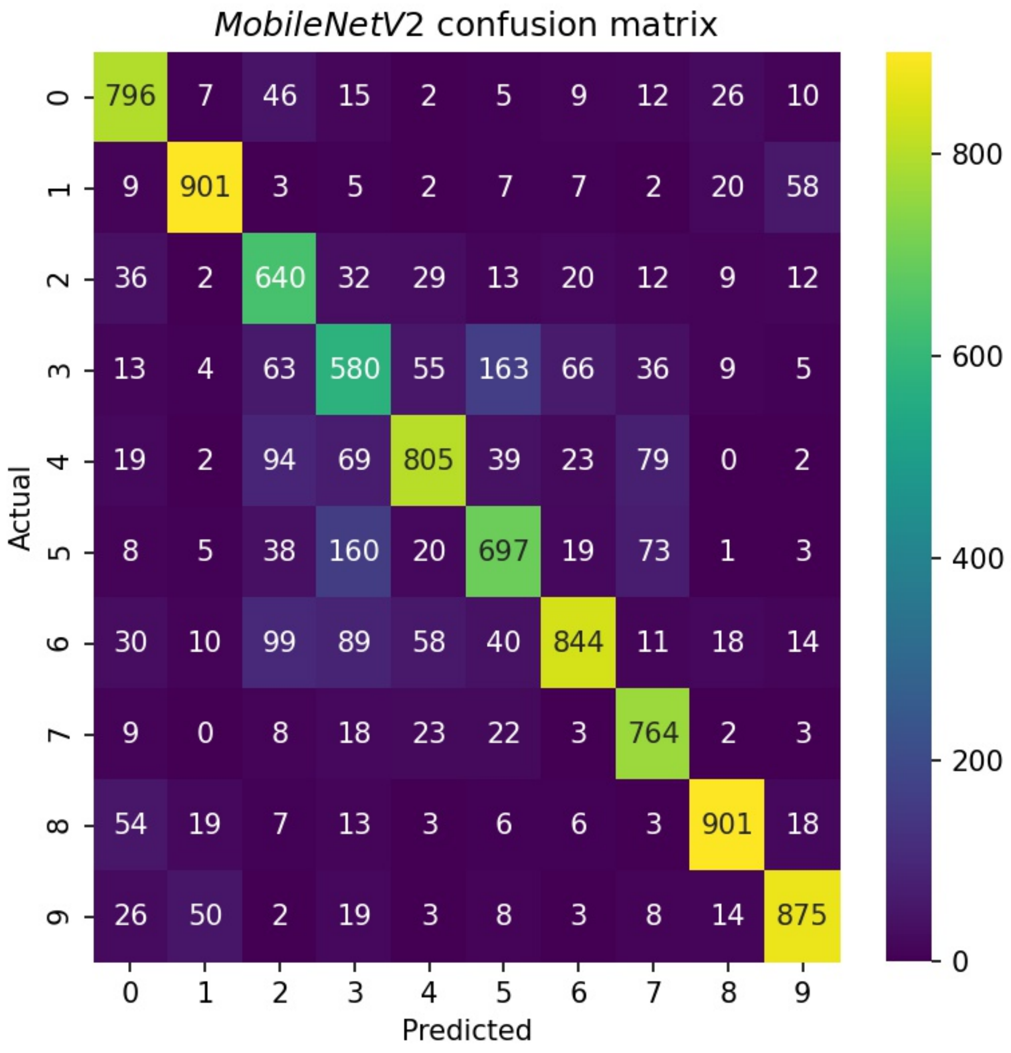


Figure 6: `MobileNetV2` confusion matrix