Deep Learning Final Presentation

Sean Graham

Part 1

Load packages

- Import the necessary items first.

```
from __future__ import absolute_import, division, print_function, unicode_literals
from collections import Counter

import os
import re
import random
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import PIL

import tensorflow as tf
import tensorflow.compat.v2 as tf
import tensorflow_datasets as tfds

from tensorflow import keras
from tensorflow.keras import datasets, layers, models
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, Flatten, Dropout, MaxPooling2D
from tensorflow.keras.preprocessing.image import ImageDataGenerator

import glob
from glob import glob

from IPython.display import display
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report

%matplotlib inline
```

Step 1: Download the dataset from Kaggle into the local disk and unzip it

- Save to “img_dir.”

```
import pathlib
img_dir = 'C:/Users/srgra/OneDrive/Documents/Deep Learning/Homework/Final Data Part 1/indoorCVPR_09/Images'
print(f'The indoorCVPR_09 photes are stored in local directory : {img_dir}')
```

```
The indoorCVPR_09 photes are stored in local directory : C:/Users/srgra/OneDrive/Documents/Deep Learning/Homework/Final Data Pa
rt 1/indoorCVPR_09/Images
```

Find number of images in each subfolder

- There are 15,619 images total.

```
total_files = 0
for root, dirs, files in os.walk(str(img_dir)):
    level = root.replace(str(img_dir), '').count(os.sep)
    indent = ' ' * 4 * (level)
    print(f'{indent}{os.path.basename(root)}/ ({len(files)} files)')
    total_files += len(files)
print(f'There are {total_files - 1} images in this dataset')
```

```
Images/ (0 files)
    airport_inside/ (608 files)
    artstudio/ (140 files)
    auditorium/ (176 files)
    bakery/ (405 files)
    bar/ (604 files)
    bathroom/ (197 files)
    bedroom/ (662 files)
    bookstore/ (380 files)
```

Print subfolder names

- There are 67 in total.

```
IndoorImage_dir = [ name for name in list(os.listdir(img_dir)) if os.path.isdir(os.path.join(img_dir, name)) ]
print(f' The Indoor Image labels = {IndoorImage_dir}')

IndoorImage_dir.sort()
print(f'\n The SORTED Indoor Image labels = {IndoorImage_dir}')

print(f'\nThere are {len(IndoorImage_dir)} classes of Indoor Images.')
```

```
 The Indoor Image labels = ['airport_inside', 'artstudio', 'auditorium', 'bakery', 'bar', 'bathroom', 'bedroom', 'bookstore',
'bowling', 'buffet', 'casino', 'children_room', 'church_inside', 'classroom', 'cloister', 'closet', 'clothingstore', 'computerr
oom', 'concert_hall', 'corridor', 'deli', 'dentaloffice', 'dining_room', 'elevator', 'fastfood_restaurant', 'florist', 'gameroo
m', 'garage', 'greenhouse', 'grocerystore', 'gym', 'hairsalon', 'hospitalroom', 'inside_bus', 'inside_subway', 'jewelleryshop',
'kindergarden', 'kitchen', 'laboratorywet', 'laundromat', 'library', 'livingroom', 'lobby', 'locker_room', 'mall', 'meeting_roo
m', 'movietheater', 'museum', 'nursery', 'office', 'operating_room', 'pantry', 'poolinside', 'prisoncell', 'restaurant', 'resta
urant_kitchen', 'shoeshop', 'stairscase', 'studiomusic', 'subway', 'toystore', 'trainstation', 'tv_studio', 'videostore', 'wait
ingroom', 'warehouse', 'winecellar']

 The SORTED Indoor Image labels = ['airport_inside', 'artstudio', 'auditorium', 'bakery', 'bar', 'bathroom', 'bedroom', 'bookst
ore', 'bowling', 'buffet', 'casino', 'children_room', 'church_inside', 'classroom', 'cloister', 'closet', 'clothingstore', 'com
puterroom', 'concert_hall', 'corridor', 'deli', 'dentaloffice', 'dining_room', 'elevator', 'fastfood_restaurant', 'florist', 'g
ameroom', 'garage', 'greenhouse', 'grocerystore', 'gym', 'hairsalon', 'hospitalroom', 'inside_bus', 'inside_subway', 'jewellery
shop', 'kindergarden', 'kitchen', 'laboratorywet', 'laundromat', 'library', 'livingroom', 'lobby', 'locker_room', 'mall', 'meet
ing_room', 'movietheater', 'museum', 'nursery', 'office', 'operating_room', 'pantry', 'poolinside', 'prisoncell', 'restaurant',
'restaurant_kitchen', 'shoeshop', 'stairscase', 'studiomusic', 'subway', 'toystore', 'trainstation', 'tv_studio', 'videostore',
'waitingroom', 'warehouse', 'winecellar']

There are 67 classes of Indoor Images.
```

Remove corrupted images

- Bad pathways were removed previously.

```
img_paths = glob(os.path.join(img_dir,'*/*.*'))

bad_paths = []

for image_path in img_paths:
    try:
        img_bytes = tf.io.read_file(image_path)
        decoded_img = tf.io.decode_image(img_bytes)
    except tf.errors.InvalidArgumentError as e:
        print(f"Found bad path {image_path}...{e}")
        bad_paths.append(image_path)
        os.remove(image_path)

print("BAD PATHS:")
for bad_path in bad_paths:
    print(f"{bad_path}")
```

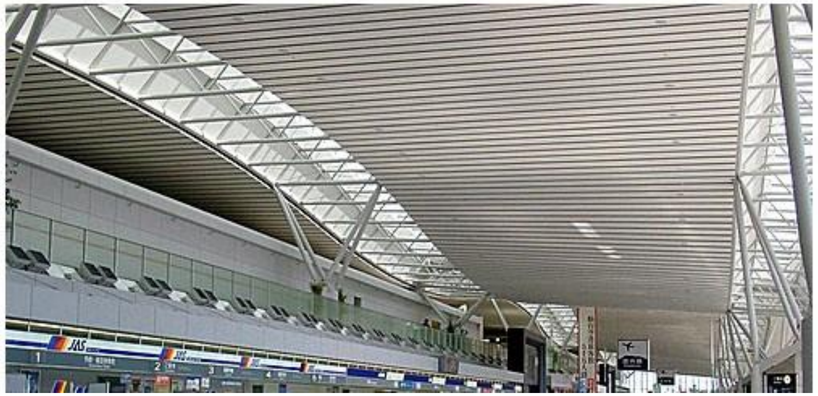
```
BAD PATHS:
```

Fix output and display sample images

- Use seed value of 777.
- Display one from each folder.

```
SEED = 777
os.environ['PYTHONHASHSEED']=str(SEED)
os.environ['TF_CUDNN_DETERMINISTIC'] = '1'
random.seed(SEED)
np.random.seed(SEED)
tf.random.set_seed(SEED)

for i in range(len(IndoorImage_dir)):
    image_file = glob(os.path.join(img_dir, IndoorImage_dir[i], '*'))
    img = PIL.Image.open(str(image_file[0]))

    print(f'(Image size  = ({img.size[0]}, {img.size[1]}, {len(img.mode)}) ; IndoorsPlace = {IndoorImage_dir[i]})')
    display(img)
```

```
(Image size  = (500, 368, 3) ; IndoorsPlace = airport_inside)
```



Set values

- Establish batch size, height, width, and split.

```
batch_size = 32
image_height = 256
image_width = 256
split = 0.2
```

Set training and validation data

- There are 15,619 training images and 3,123 validation images.

```
train_data = tf.keras.preprocessing.image_dataset_from_directory(
  img_dir,
  labels='inferred',
  label_mode='int',
  validation_split= split,
  subset="training",
  seed= 1001,
  image_size=(image_height, image_width),
  batch_size=batch_size)
```

```
Found 15619 files belonging to 67 classes.
Using 12496 files for training.
```

```
val_data = tf.keras.preprocessing.image_dataset_from_directory(
  img_dir,
  labels='inferred',
  label_mode='int',
  validation_split= split,
  subset="validation",
  seed=1001,
  image_size=(image_height, image_width),
  batch_size=batch_size)
```

```
Found 15619 files belonging to 67 classes.
Using 3123 files for validation.
```

Visualize as matrices

```
for img, lab in train_data.take(1):
    print(img[1].numpy().astype("uint16"))
    print(f'minimum = {np.amin(img[0].numpy().astype("uint16"))}, maximum = {np.amax(img[0].numpy().astype("uint16"))}')
    break
```

```
[[[236 225 203]
  [254 248 226]
  [252 249 232]
  ...
  [219 171 109]
  [203 170 129]
  [255 250 220]]

 [[240 229 209]
  [255 248 229]
  [249 248 230]
  ...
  [225 177 115]
  [207 174 133]
```

Plot 4x4 sample of images

- Use appropriate labels.

```
plt.figure(figsize=(12, 12))

for img, lab in train_data.take(1):
    for i in range(16):
        ax = plt.subplot(4, 4, i + 1)
        plt.imshow(img[i].numpy().astype("uint16"))
        plt.title(IndoorImage_dir[lab[i]])
        plt.axis("off")
```

```python
for image_batch, labels_batch in train_data:
    print(f'image_batch.shape = {image_batch.shape} \nlabels_batch.shape = {labels_batch.shape } ')
    break
```

```
image_batch.shape = (32, 256, 256, 3)
labels_batch.shape = (32,)
```

```python
AUTOTUNE = tf.data.AUTOTUNE

train_data = train_data.cache().shuffle(1000).prefetch(buffer_size=AUTOTUNE)
val_data = val_data.cache().prefetch(buffer_size=AUTOTUNE)
```

Double check parameters and configure the dataset

- Apply caching, shuffle, and prefetch.

Step 2: Build a baseline convolutional neural network on the training dataset and evaluate it on the test dataset

Add to model layer by layer.

```
model = Sequential([
  layers.experimental.preprocessing.Rescaling(1./255, input_shape=(image_height, image_width, 3)),
  layers.Conv2D(32, (3, 3), padding='same', activation='relu'),
  layers.MaxPooling2D(2,2),
  layers.Conv2D(64, (3, 3), activation='relu'),
  layers.MaxPooling2D((2,2)),
  layers.Conv2D(64, (3, 3), activation='relu'),
  layers.MaxPooling2D((2,2)),
  layers.Flatten(),
  layers.Dense(64, activation='relu'),
  layers.Dense(67)
])
```

Summarize the model

```
model.summary()

Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling (Rescaling)       (None, 256, 256, 3)       0

 conv2d (Conv2D)             (None, 256, 256, 32)      896

 max_pooling2d (MaxPooling2D  (None, 128, 128, 32)     0
 )

 conv2d_1 (Conv2D)           (None, 126, 126, 64)      18496

 max_pooling2d_1 (MaxPooling  (None, 63, 63, 64)       0
 2D)

 conv2d_2 (Conv2D)           (None, 61, 61, 64)        36928

 max_pooling2d_2 (MaxPooling  (None, 30, 30, 64)       0
 2D)

 flatten (Flatten)           (None, 57600)             0
```

Configure the model

```
model.compile(optimizer='adam', loss=tf.keras.losses.
SparseCategoricalCrossentropy(from_logits=True), metrics=['accuracy'])
```

Train the model

- Epochs are set to 5 and patience to 3.

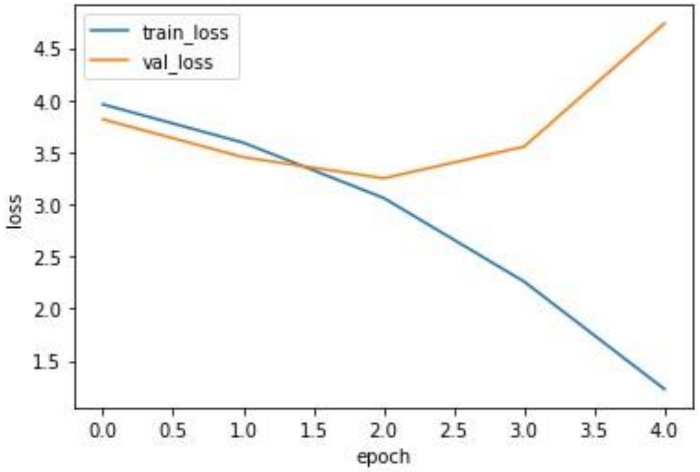
```
%%time

callback = tf.keras.callbacks.EarlyStopping(monitor = 'val_accuracy', patience = 3)
history = model.fit(train_data, validation_data = val_data, epochs = 5, callbacks = [callback], verbose = 1)
```

```
Epoch 1/5
391/391 [==============================] - 818s 2s/step - loss: 3.9596 - accuracy: 0.0649 - val_loss: 3.8156 - val_accuracy: 0.
0810
Epoch 2/5
391/391 [==============================] - 747s 2s/step - loss: 3.5916 - accuracy: 0.1244 - val_loss: 3.4525 - val_accuracy: 0.
1511
Epoch 3/5
391/391 [==============================] - 776s 2s/step - loss: 3.0611 - accuracy: 0.2258 - val_loss: 3.2499 - val_accuracy: 0.
1876
Epoch 4/5
391/391 [==============================] - 768s 2s/step - loss: 2.2587 - accuracy: 0.4011 - val_loss: 3.5535 - val_accuracy: 0.
1899
Epoch 5/5
391/391 [==============================] - 771s 2s/step - loss: 1.2265 - accuracy: 0.6569 - val_loss: 4.7369 - val_accuracy: 0.
1761
CPU times: total: 5h 59min 23s
Wall time: 1h 4min 41s
```

Make plots

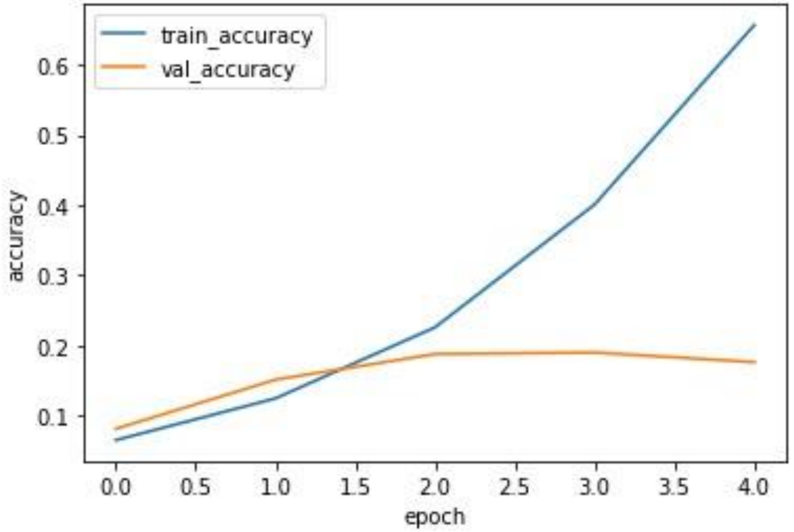
```python
train_history = pd.DataFrame(history.history)
train_history['epoch'] = history.epoch

sns.lineplot(x='epoch', y ='loss', data = train_history)
sns.lineplot(x='epoch', y ='val_loss', data = train_history)
plt.legend(labels=['train_loss', 'val_loss'])
```

```
<matplotlib.legend.Legend at 0x2000dfae6a0>
```




```python
sns.lineplot(x='epoch', y ='accuracy', data =train_history)
sns.lineplot(x='epoch', y ='val_accuracy', data =train_history)
plt.legend(labels=['train_accuracy', 'val_accuracy'])
```

```
<matplotlib.legend.Legend at 0x2000bc98cd0>
```



Run classification report

```
y_pred_prob = model.predict(img)
score = tf.nn.softmax(y_pred_prob)
y_pred = np.argmax(score, axis = 1)
print(classification_report (lab, y_pred))
```

```
    accuracy                           0.94        32
   macro avg       0.95      0.92      0.93        32
weighted avg       0.98      0.94      0.94        32
```

Step 3: Build a second CNN model with data augmentation and dropout layers

- Apply transformations to images.

```
data_aug = tf.keras.Sequential([tf.keras.layers.experimental.preprocessing.RandomFlip("horizontal_and_vertical",
    input_shape=(image_height, image_width, 3)),
    tf.keras.layers.experimental.preprocessing.RandomRotation(0.1),
    tf.keras.layers.experimental.preprocessing.RandomTranslation(height_factor=0.1, width_factor = 0.1),
    tf.keras.layers.experimental.preprocessing.RandomZoom(height_factor=(0.1, 0.1))])
```

Normalize images and print sample

- Fix height and width, divide by 255, and produce 16 images.

```
def normalize_image(image, label, target_height = 256, target_width = 256):
    image = tf.cast(image, tf.float32)/255.
    image = tf.image.resize_with_crop_or_pad(image, target_height, target_width)
    return image, label
train_data_normalized = train_data.map(normalize_image, num_parallel_calls = tf.data.experimental.AUTOTUNE)

plt.figure(figsize=(12, 12))
for image, label in train_data_normalized.take(1):
    for i in range(16):
        aug_images = data_aug(image)
        ax = plt.subplot(4, 4, i + 1)
        plt.imshow(aug_images[0])
        plt.axis("off")
```

Build the model

- Build layers with necessary changes.

```
model = Sequential([data_aug,
  layers.experimental.preprocessing.Rescaling(1./255, input_shape=(image_height, image_width, 3)),
  layers.Conv2D(32, (3, 3), padding='same', activation='relu'),
  layers.MaxPooling2D(2,2),
  layers.Conv2D(64, (3, 3), activation='relu'),
  layers.MaxPooling2D((2,2)),
  layers.Dropout(0.25),
  layers.Flatten(),
  layers.Dense(64, activation='relu'),
  layers.Dropout(0.25),
  layers.Dense(67)
])
```

Summary

```
model.summary()

Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 sequential (Sequential)     (None, 256, 256, 3)       0

 rescaling (Rescaling)       (None, 256, 256, 3)       0

 conv2d (Conv2D)             (None, 256, 256, 32)      896

 max_pooling2d (MaxPooling2D  (None, 128, 128, 32)     0
 )

 conv2d_1 (Conv2D)           (None, 126, 126, 64)      18496

 max_pooling2d_1 (MaxPooling  (None, 63, 63, 64)       0
 2D)

 dropout (Dropout)           (None, 63, 63, 64)        0

 flatten (Flatten)           (None, 254016)            0

 dense (Dense)               (None, 64)                16257088

 dropout_1 (Dropout)         (None, 64)                0

 dense_1 (Dense)             (None, 67)                4355
```

Model configuration

```
model.compile(optimizer='adam', loss=tf.keras.losses.
SparseCategoricalCrossentropy(from_logits=True), metrics=['accuracy'])
```

Training

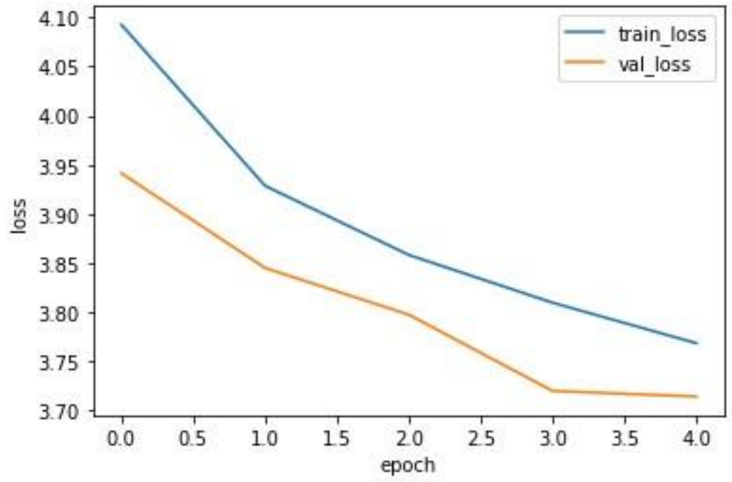
```
%%time

callback = tf.keras.callbacks.EarlyStopping(monitor='val_accuracy', patience= 3)
history = model.fit(train_data, epochs=5, validation_data=(val_data), callbacks=[callback], verbose = 1)
```

```
Epoch 1/5
391/391 [==============================] - 813s 2s/step - loss: 4.0926 - accuracy: 0.0523 - val_loss: 3.9412 - val_accuracy: 0.
0746
Epoch 2/5
391/391 [==============================] - 790s 2s/step - loss: 3.9285 - accuracy: 0.0670 - val_loss: 3.8448 - val_accuracy: 0.
0967
Epoch 3/5
391/391 [==============================] - 791s 2s/step - loss: 3.8580 - accuracy: 0.0766 - val_loss: 3.7972 - val_accuracy: 0.
0970
Epoch 4/5
391/391 [==============================] - 782s 2s/step - loss: 3.8094 - accuracy: 0.0810 - val_loss: 3.7195 - val_accuracy: 0.
1053
Epoch 5/5
391/391 [==============================] - 778s 2s/step - loss: 3.7682 - accuracy: 0.0843 - val_loss: 3.7138 - val_accuracy: 0.
0938
CPU times: total: 6h 14min 45s
Wall time: 1h 5min 54s
```

Make plots

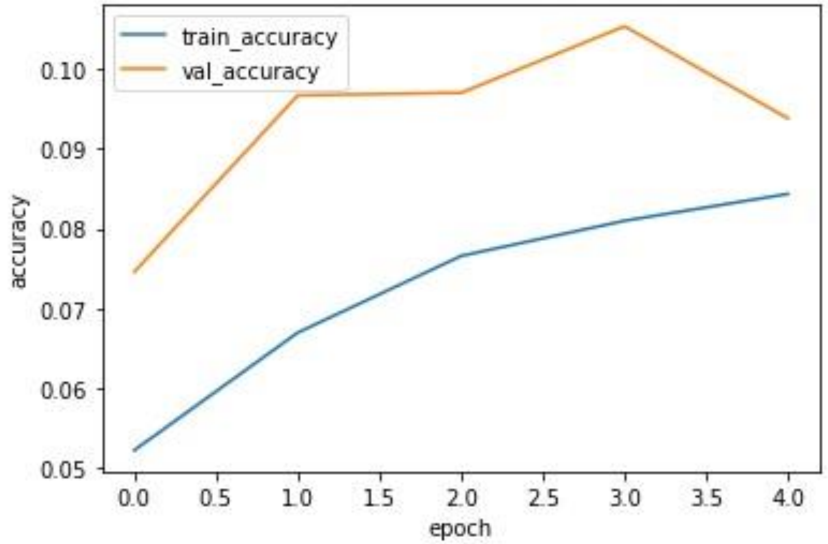
```
train_history = pd.DataFrame(history.history)
train_history['epoch'] = history.epoch

sns.lineplot(x='epoch', y ='loss', data = train_history)
sns.lineplot(x='epoch', y ='val_loss', data = train_history)
plt.legend(labels=['train_loss', 'val_loss'])
```

```
<matplotlib.legend.Legend at 0x1ae211c9eb0>
```




```
sns.lineplot(x='epoch', y ='accuracy', data =train_history)
sns.lineplot(x='epoch', y ='val_accuracy', data =train_history)
plt.legend(labels=['train_accuracy', 'val_accuracy'])
```

```
<matplotlib.legend.Legend at 0x1ae04cd4460>
```



Classification report

```
y_pred_prob = model.predict(img)
score = tf.nn.softmax(y_pred_prob)
y_pred = np.argmax(score, axis = 1)
print(classification_report (lab, y_pred))
```

```
    accuracy                           0.16        32
   macro avg       0.08      0.09      0.07        32
weighted avg       0.16      0.16      0.13        32
```

Step 4: Build a CNN model based on a pre-trained model

Continue to use the same parameters.

```
IMG_SHAPE = (image_height, image_width, 3)

MobileNetV3Large_model = tf.keras.applications.MobileNetV3Large(input_shape = IMG_SHAPE, include_top=False, weights='imagenet')
```

```
WARNING:tensorflow:`input_shape` is undefined or non-square, or `rows` is not 224. Weights for input shape (224, 224) will be l
oaded as the default.
```

Run summary

```
MobileNetV3Large_model.summary()
```

```
Model: "MobilenetV3large"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_1 (InputLayer)           [(None, 256, 256, 3  0           []
                                )]

 rescaling_1 (Rescaling)        (None, 256, 256, 3)  0           ['input_1[0][0]']

 Conv (Conv2D)                  (None, 128, 128, 16  432         ['rescaling_1[0][0]']
                                )

 Conv/BatchNorm (BatchNormaliza  (None, 128, 128, 16  64         ['Conv[0][0]']
 tion)                          )

 tf.__operators__.add (TFOpLamb  (None, 128, 128, 16  0          ['Conv/BatchNorm[0][0]']
 da)                            )

 re_lu (ReLU)                   (None, 128, 128, 16  0           ['tf.__operators__.add[0][0]']
```

Freeze convolutional base and import function from model

- Ensure the weights aren't updated.

```
MobileNetV3Large_model.trainable = False

preprocess_input = tf.keras.applications.mobilenet_v3.preprocess_input
```

Adjust dimensions

- Convert 4D to 2D using GlobalAveragePooling2D.

```
image_batch, label_batch = next(iter(train_data))
feature_batch = MobileNetV3Large_model(image_batch)
print(feature_batch.shape)
```

```
(32, 1, 1, 1280)
```

```
global_average_layer = tf.keras.layers.GlobalAveragePooling2D()
feature_batch_average = global_average_layer(feature_batch)
print(feature_batch_average.shape)
```

```
(32, 1280)
```

Customize top layer

- Outputs a 32x67 matrix.

```
prediction_layer = tf.keras.layers.Dense(67)
prediction_batch = prediction_layer(feature_batch_average)
print(f' The size of the predicted value for a given batch = {prediction_batch.shape}')
print(prediction_batch)
```

```
 The size of the predicted value for a given batch = (32, 67)
tf.Tensor(
[[-2.0409122e-01  6.3577390e-01  2.3614883e-02 ...  7.2075760e-01
   1.8778598e-01 -1.7353135e-01]
 [-2.6219270e-01  1.5747998e+00  2.7948743e-01 ... -3.6154410e-01
  -2.4913867e+00  3.8727441e-01]
 [ 1.4375815e-01 -1.6904128e-01  5.5342245e-01 ... -5.1366943e-01
  -2.0877447e+00  1.1886594e+00]
 ...
 [-9.4741005e-01 -6.7172086e-01 -7.0315319e-01 ...  4.2533940e-01
  -2.4297982e-03 -1.2049288e-02]
 [ 6.0112774e-01 -9.2111111e-01  1.0486938e+00 ...  1.7056112e+00
   4.0607533e-01 -2.8523833e-02]
 [ 7.4466980e-01  1.1435473e+00  5.5339587e-01 ... -3.5331130e-02
  -5.2686000e-01 -5.9807584e-02]], shape=(32, 67), dtype=float32)
```

Create model using transfer learning

- Add the layers that have been created.

```
inputs = tf.keras.Input(shape = IMG_SHAPE)

x = data_aug(inputs)
x = preprocess_input(x)

x = MobileNetV3Large_model(x, training=False)

x = global_average_layer(x)
x = tf.keras.layers.Dropout(0.2)(x)

outputs = prediction_layer(x)

model = tf.keras.Model(inputs, outputs)
```

Configure model

- Classes are encoded as integers.

```
learning_rate = 0.0001
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=learning_rate),
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

Run summary

```
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 256, 256, 3)]     0

 sequential (Sequential)     (None, 256, 256, 3)       0

 MobilenetV3large (Functiona  (None, 1, 1, 1280)       4226432
 l)

 global_average_pooling2d_1   (None, 1280)             0
 (GlobalAveragePooling2D)

 dropout_2 (Dropout)         (None, 1280)              0

 dense_2 (Dense)             (None, 67)                85827

=================================================================
Total params: 4,312,259
Trainable params: 85,827
Non-trainable params: 4,226,432
_________________________________________________________________
```

Train the model

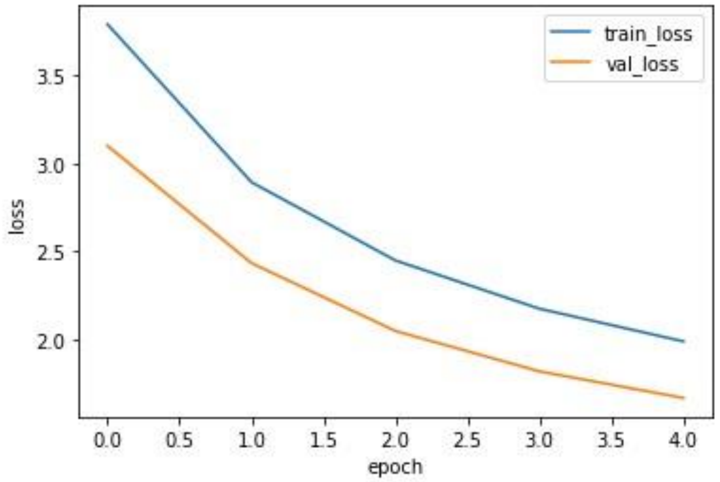
```
%%time

callback = tf.keras.callbacks.EarlyStopping(monitor='val_accuracy', patience= 3)
history = model.fit(train_data, epochs=5, validation_data=(val_data), callbacks=[callback], verbose = 1)
```

```
Epoch 1/5
391/391 [==============================] - 695s 2s/step - loss: 3.7904 - accuracy: 0.1159 - val_loss: 3.0996 - val_accuracy: 0.
3020
Epoch 2/5
391/391 [==============================] - 695s 2s/step - loss: 2.8925 - accuracy: 0.2844 - val_loss: 2.4326 - val_accuracy: 0.
4441
Epoch 3/5
391/391 [==============================] - 695s 2s/step - loss: 2.4473 - accuracy: 0.3818 - val_loss: 2.0461 - val_accuracy: 0.
5203
Epoch 4/5
391/391 [==============================] - 696s 2s/step - loss: 2.1729 - accuracy: 0.4360 - val_loss: 1.8166 - val_accuracy: 0.
5572
Epoch 5/5
391/391 [==============================] - 693s 2s/step - loss: 1.9871 - accuracy: 0.4806 - val_loss: 1.6653 - val_accuracy: 0.
5799
CPU times: total: 5h 4min 40s
Wall time: 57min 54s
```

Make plots

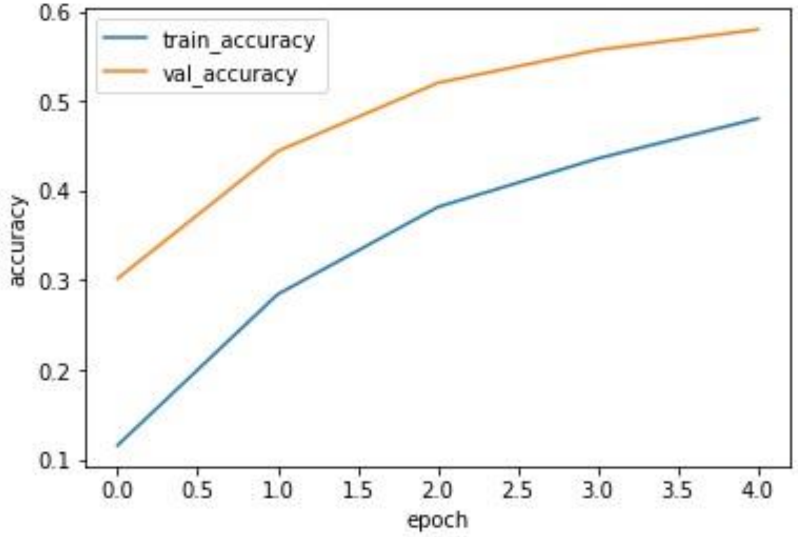
```
train_history = pd.DataFrame(history.history)
train_history['epoch'] = history.epoch

sns.lineplot(x='epoch', y ='loss', data = train_history)
sns.lineplot(x='epoch', y ='val_loss', data = train_history)
plt.legend(labels=['train_loss', 'val_loss'])
```

```
<matplotlib.legend.Legend at 0x1ae38dfda00>
```




```
sns.lineplot(x='epoch', y ='accuracy', data =train_history)
sns.lineplot(x='epoch', y ='val_accuracy', data =train_history)
plt.legend(labels=['train_accuracy', 'val_accuracy'])
```

```
<matplotlib.legend.Legend at 0x1ae620b89a0>
```



Classification Report

```
y_pred_prob = model.predict(img)
score = tf.nn.softmax(y_pred_prob)
y_pred = np.argmax(score, axis = 1)
print(classification_report (lab, y_pred))
```

```
    accuracy                           0.59        32
   macro avg       0.34      0.38      0.36        32
weighted avg       0.54      0.59      0.56        32
```

Step 5: Which of the three models is best?

- Question 2 has the best model; the accuracy is highest.

Part 2

Load Packages

Load necessary items first.

```
import pandas as pd
from sklearn import preprocessing
import re
from sklearn.model_selection import train_test_split
import tensorflow as tf
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, classification_report
```

Step 1: Download the dataset from Kaggle into the local disk and unzip it

Step 2: Clean and preprocess the text data and split it into training and test data

```
review_file_path = 'C:/Users/srgra/OneDrive/Documents/Deep Learning/Homework/Final Data Part 2/Restaurant_Reviews.tsv'
```

```
review_tsv_ds = pd.read_csv(review_file_path, sep = '\t')
review_tsv_ds.head()
```

	Review	Liked
0	Wow... Loved this place.	1
1	Crust is not good.	0
2	Not tasty and the texture was just nasty.	0
3	Stopped by during the late May bank holiday of...	1
4	The selection on the menu was great and so wer...	1

```
review_tsv_ds.Liked.value_counts()
```

```
1    500
0    500
Name: Liked, dtype: int64
```

Remove special characters, punctuation, spaces, and words shorter than 2 letters

```
review_tsv_ds['Review'] = review_tsv_ds['Review'].apply(lambda x: re.sub(r'[^A-Za-z0-9]+',' ',x))
review_tsv_ds['Review'] = review_tsv_ds['Review'].apply(lambda x: re.sub(r"<br />", " ", x))
review_tsv_ds['Review'] = review_tsv_ds['Review'].apply(lambda x: re.sub(r'\b[a-zA-Z]{1,2}\b', '', x))
review_tsv_ds.head()
```

	Review	Liked
0	Wow Loved this place	1
1	Crust not good	0
2	Not tasty and the texture was just nasty	0
3	Stopped during the late May bank holiday off ...	1
4	The selection the menu was great and were th...	1

Split the data, turn into numerical values, perform text vectorization, and fit layer

- Specify a vocabulary size of 1000.

```
X = review_tsv_ds['Review'].values
y = review_tsv_ds['Liked'].values
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify = y, test_size = 0.2)
```

```
print(f'X_train size  = {X_train.shape}; X_test size  = {X_test.shape}')
```

```
X_train size  = (800,); X_test size  = (200,)
```

```
VOCAB_SIZE = 1000
encoder = tf.keras.layers.experimental.preprocessing.TextVectorization(max_tokens=VOCAB_SIZE)
encoder.adapt(X_train)
```

Step 3: Build a baseline RNN using and embedding layer and GRU

```
model = tf.keras.Sequential([
    encoder,
    tf.keras.layers.Embedding(
        input_dim=len(encoder.get_vocabulary()),
        output_dim=64,
        mask_zero=True),
    tf.keras.layers.GRU(128, return_sequences = False),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(1)
])
```

Compile and train the model

```
model.compile(loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
              optimizer=tf.keras.optimizers.Adam(1e-4),
              metrics=['accuracy'])
```

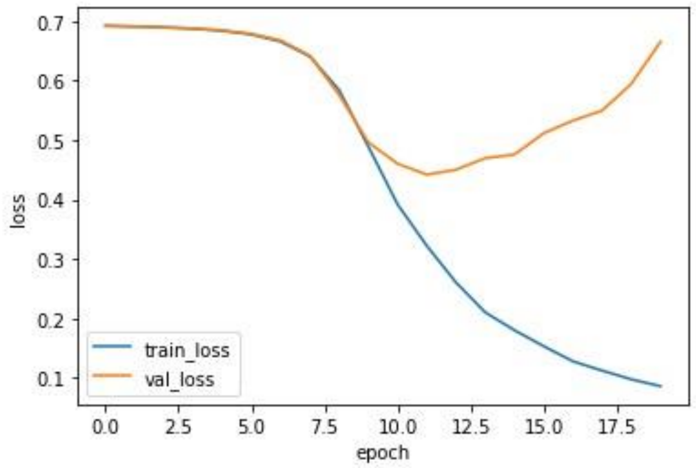
```
%%time
history = model.fit(x=X_train, y=y_train, batch_size= 32, epochs= 20,
          validation_data=(X_test,y_test), verbose= 1)
```

```
Epoch 1/20
25/25 [==============================] - 8s 103ms/step - loss: 0.6926 - accuracy: 0.5000 - val_loss: 0.6917 - val_accuracy:
0.5000
Epoch 2/20
25/25 [==============================] - 1s 20ms/step - loss: 0.6912 - accuracy: 0.5000 - val_loss: 0.6909 - val_accuracy: 0.
5000
Epoch 3/20
25/25 [==============================] - 1s 20ms/step - loss: 0.6897 - accuracy: 0.5000 - val_loss: 0.6897 - val_accuracy: 0.
5000
Epoch 4/20
25/25 [==============================] - 1s 26ms/step - loss: 0.6875 - accuracy: 0.5000 - val_loss: 0.6878 - val_accuracy: 0.
5000
Epoch 5/20
25/25 [==============================] - 1s 21ms/step - loss: 0.6839 - accuracy: 0.5000 - val_loss: 0.6847 - val_accuracy: 0.
5000
Epoch 6/20
25/25 [==============================] - 0s 19ms/step - loss: 0.6778 - accuracy: 0.5000 - val_loss: 0.6786 - val_accuracy: 0
```

Make plots

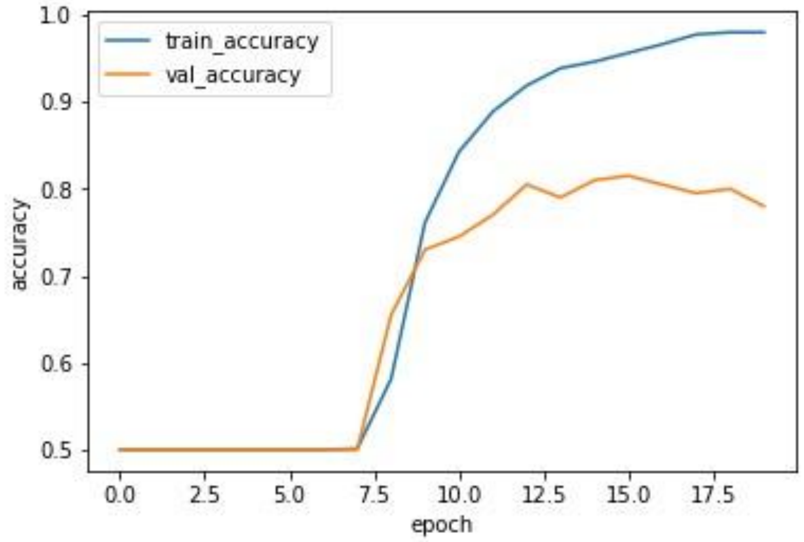
```
train_history = pd.DataFrame(history.history)
train_history['epoch'] = history.epoch

sns.lineplot(x='epoch', y ='loss', data =train_history)
sns.lineplot(x='epoch', y ='val_loss', data =train_history)
plt.legend(labels=['train_loss', 'val_loss'])
```

```
<matplotlib.legend.Legend at 0x1d625f9ae50>
```




```
sns.lineplot(x='epoch', y ='accuracy', data =train_history)
sns.lineplot(x='epoch', y ='val_accuracy', data =train_history)
plt.legend(labels=['train_accuracy', 'val_accuracy'])
```

```
<matplotlib.legend.Legend at 0x1d60fa87fd0>
```



Forecast labels and get confusion matrix

- For the confusion matrix, (79 + 77) / (79 + 21 + 23 + 77) = 0.78.

```
y_pred = (model.predict(X_test) > 0.5).astype(int)
```

```
confusion_matrix(y_test, y_pred)
```

```
array([[79, 21],
       [23, 77]], dtype=int64)
```

Classification report

```
label_names = ['negative', 'positive']
print(classification_report(y_test, y_pred, target_names=label_names))
```

```
              precision    recall  f1-score   support

    negative       0.77      0.79      0.78       100
    positive       0.79      0.77      0.78       100

    accuracy                           0.78       200
   macro avg       0.78      0.78      0.78       200
weighted avg       0.78      0.78      0.78       200
```

Step 4: build a RNN using embedding and LSTM

```
model = tf.keras.Sequential([
    encoder,
    tf.keras.layers.Embedding(
        input_dim=len(encoder.get_vocabulary()),
        output_dim=64,
        mask_zero=True),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(128)),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(1)
])
```

Compile and train the model

```
model.compile(loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
              optimizer=tf.keras.optimizers.Adam(1e-4),
              metrics=['accuracy'])
```

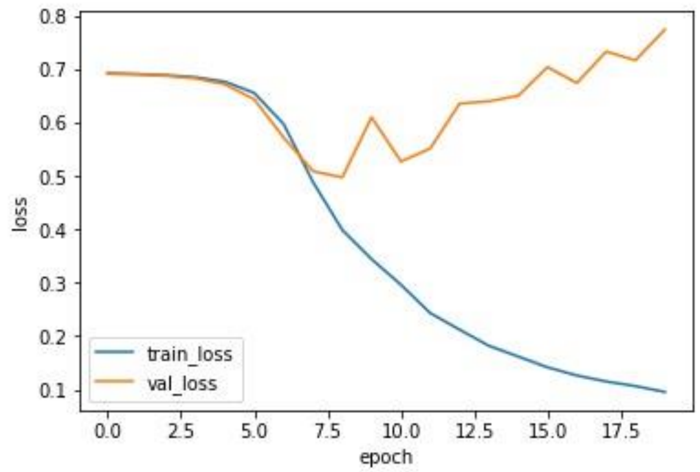
```
%%time
history = model.fit(x=X_train, y=y_train, batch_size= 32, epochs= 20,
          validation_data=(X_test,y_test), verbose= 1)
```

```
Epoch 1/20
25/25 [==============================] - 13s 165ms/step - loss: 0.6926 - accuracy: 0.5000 - val_loss: 0.6918 - val_accuracy: 0.
5000
Epoch 2/20
25/25 [==============================] - 1s 35ms/step - loss: 0.6912 - accuracy: 0.5000 - val_loss: 0.6905 - val_accuracy: 0.50
00
Epoch 3/20
25/25 [==============================] - 1s 38ms/step - loss: 0.6892 - accuracy: 0.5000 - val_loss: 0.6878 - val_accuracy: 0.50
00
Epoch 4/20
25/25 [==============================] - 1s 37ms/step - loss: 0.6852 - accuracy: 0.5000 - val_loss: 0.6830 - val_accuracy: 0.50
00
Epoch 5/20
25/25 [==============================] - 1s 36ms/step - loss: 0.6768 - accuracy: 0.5000 - val_loss: 0.6718 - val_accuracy: 0.50
00
Epoch 6/20
25/25 [==============================] - 1s 34ms/step - loss: 0.6558 - accuracy: 0.5000 - val_loss: 0.6438 - val_accuracy: 0.50
```

Make plots

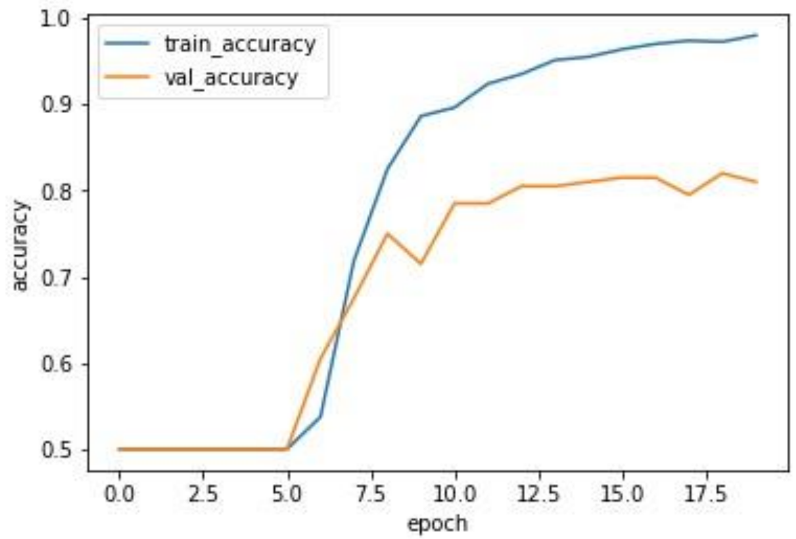
```
train_history = pd.DataFrame(history.history)
train_history['epoch'] = history.epoch

sns.lineplot(x='epoch', y ='loss', data =train_history)
sns.lineplot(x='epoch', y ='val_loss', data =train_history)
plt.legend(labels=['train_loss', 'val_loss'])
```

```
<matplotlib.legend.Legend at 0x1d62c055b50>
```




```
sns.lineplot(x='epoch', y ='accuracy', data =train_history)
sns.lineplot(x='epoch', y ='val_accuracy', data =train_history)
plt.legend(labels=['train_accuracy', 'val_accuracy'])
```

```
<matplotlib.legend.Legend at 0x1d635a5d610>
```



Forecast labels and confusion matrix

- For the confusion matrix, (84 + 78) / (84 + 16 + 22 + 78) = 0.81.

```
y_pred = (model.predict(X_test) > 0.5).astype(int)
```

```
confusion_matrix(y_test, y_pred)
```

```
array([[84, 16],
       [22, 78]], dtype=int64)
```

Classification report

```
label_names = ['negative', 'positive']
print(classification_report(y_test, y_pred, target_names=label_names))
```

```
              precision    recall  f1-score   support

    negative       0.79      0.84      0.82       100
    positive       0.83      0.78      0.80       100

    accuracy                           0.81       200
   macro avg       0.81      0.81      0.81       200
weighted avg       0.81      0.81      0.81       200
```

Step 5: Build a RNN model using embedding, GRU, and LSTM

```
model = tf.keras.Sequential([
    encoder,
    tf.keras.layers.Embedding(
        input_dim=len(encoder.get_vocabulary()),
        output_dim=64,
        mask_zero=True),
    tf.keras.layers.GRU(128, return_sequences=True),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(128)),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(1)
])
```

Compile and train model

```
model.compile(loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
              optimizer=tf.keras.optimizers.Adam(1e-4),
              metrics=['accuracy'])
```

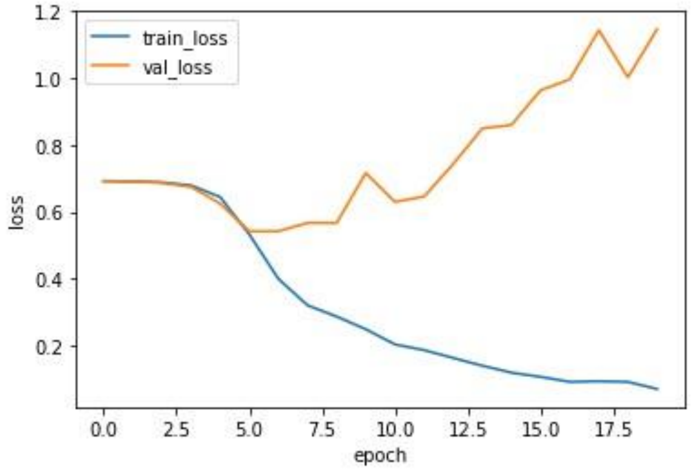
```
%%time
history = model.fit(x=X_train, y=y_train, batch_size= 32, epochs= 20,
          validation_data=(X_test,y_test), verbose= 1)
```

```
Epoch 1/20
25/25 [==============================] - 18s 244ms/step - loss: 0.6929 - accuracy: 0.5000 - val_loss: 0.6924 - val_accuracy: 0.
5000
Epoch 2/20
25/25 [==============================] - 1s 57ms/step - loss: 0.6916 - accuracy: 0.5000 - val_loss: 0.6911 - val_accuracy: 0.50
00
Epoch 3/20
25/25 [==============================] - 1s 58ms/step - loss: 0.6888 - accuracy: 0.5000 - val_loss: 0.6873 - val_accuracy: 0.50
00
Epoch 4/20
25/25 [==============================] - 1s 58ms/step - loss: 0.6797 - accuracy: 0.5000 - val_loss: 0.6750 - val_accuracy: 0.50
00
Epoch 5/20
25/25 [==============================] - 2s 62ms/step - loss: 0.6448 - accuracy: 0.5050 - val_loss: 0.6253 - val_accuracy: 0.51
00
Epoch 6/20
25/25 [==============================] - 1s 59ms/step - loss: 0.5331 - accuracy: 0.6600 - val_loss: 0.5423 - val_accuracy: 0.66
```

Make Plots

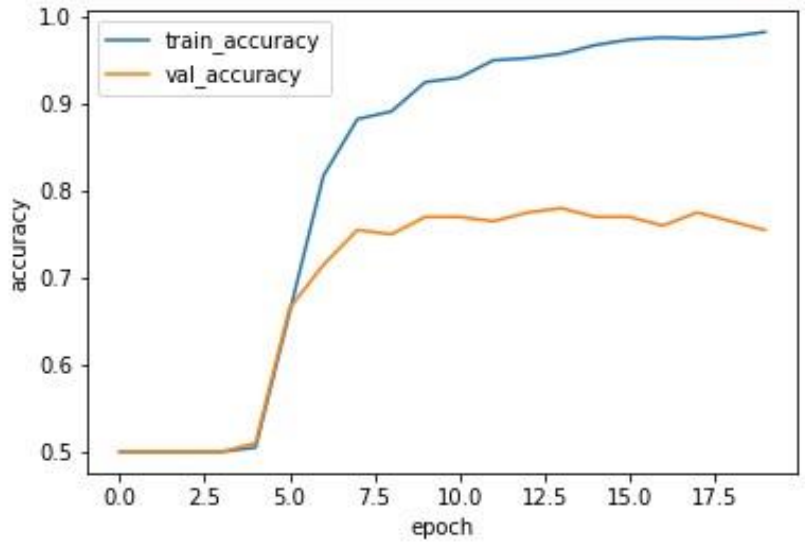
```
train_history = pd.DataFrame(history.history)
train_history['epoch'] = history.epoch

sns.lineplot(x='epoch', y ='loss', data =train_history)
sns.lineplot(x='epoch', y ='val_loss', data =train_history)
plt.legend(labels=['train_loss', 'val_loss'])
```

```
<matplotlib.legend.Legend at 0x1d637d1e3d0>
```




```
sns.lineplot(x='epoch', y ='accuracy', data =train_history)
sns.lineplot(x='epoch', y ='val_accuracy', data =train_history)
plt.legend(labels=['train_accuracy', 'val_accuracy'])
```

```
<matplotlib.legend.Legend at 0x1d64d133d60>
```



Forecast labels and confusion matrix

- For the confusion matrix, (78 + 73) / (78 + 22 + 27 + 73) = 0.76.

```
y_pred = (model.predict(X_test) > 0.5).astype(int)
```

```
confusion_matrix(y_test, y_pred)
```

```
array([[78, 22],
       [27, 73]], dtype=int64)
```

Classification Report

```
label_names = ['negative', 'positive']
print(classification_report(y_test, y_pred, target_names=label_names))
```

```
              precision    recall  f1-score   support

    negative       0.74      0.78      0.76       100
    positive       0.77      0.73      0.75       100

    accuracy                           0.76       200
   macro avg       0.76      0.76      0.75       200
weighted avg       0.76      0.76      0.75       200
```

Step 6: Which of the three models is best?

- Question 4 has the best model; the accuracy is highest.
- Running again may produce different results.

Questions?