1. Overview

- Objective: To build an image classification model to recognize and classify 10 different types of food

- Universal workflow of machine learning framework
  - Step 1: Defining the problem and assembling a dataset
  - Step 2: Choosing a measure of success
  - Step 3: Deciding on an evaluation protocol
  - Step 4: Preparing your data
  - Step 5: Developing a model that does better than a baseline
  - Step 6: Scaling up: train the model until it overfits
  - Step 7: Regularizing your model and tuning your hyperparameters

1. OVERVIEW

STEP 1: DEFINING THE PROBLEM AND ASSEMBLING A DATASET

- Type of Problem

  A multiclass classification problem with 10 classes of output

- Inputs and Outputs
  - Inputs (training): 750 food images per type
  - Inputs (validation): 200 food images per type
  - Inputs (testing): 50 food images per type
  - Output: Food labels

1. Overview

Step 2: Choosing a measure of success
Step 3: Deciding on an evaluation protocol

- <u>Measure of Success</u>

  Accuracy

- <u>Evaluation Protocol</u>

  Maintaining a hold-out validation set

  (Dataset is huge – images)

2. Data Preprocessing and Data Loading

Step 4: Prepare your data

- Platform
  - Google Drive
    - Issues encountered during uploading of photos. Training folder had more images than it should have.
  - Google Colab
- Data Loading

```
import os
# Load the Drive helper and mount
from google.colab import drive

# This will prompt for authorization.
drive.mount('/content/drive')
# After executing the cell above, Drive
# files will be present in "/content/drive/My Drive".
!ls "/content/drive/My Drive/"

base_dir = "/content/drive/My Drive/NP DL/ASG1/food"

train_dir = os.path.join(base_dir, 'train')
validation_dir = os.path.join(base_dir, 'validation')
test_dir = os.path.join(base_dir, 'test')
```

Linking Colab to Google Drive

Folder directory of training, validation and testing dataset

2. DATA PREPROCESSING AND DATA LOADING

- Import package

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

- Data Preprocessing

```
# data preprocessing

# rescale pixel values (0 and 255) to [0, 1] interval
train_datagen = ImageDataGenerator(rescale=1./255)  # floating point
test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
        # path to target directory
        train_dir,
        # images target size
        target_size=(img_size, img_size),
        batch_size=75,
        # type of label arrays
        class_mode='categorical')

validation_generator = test_datagen.flow_from_directory(
        validation_dir,
        target_size=(img_size, img_size),
        batch_size=40,
        class_mode='categorical')
```

```
Found 7500 images belonging to 10 classes.
Found 2000 images belonging to 10 classes.
```

Rescale images to [0, 1] by dividing by 255 (0: black; 255: white)

Image size standardised to 150 x 150 px

```
# image size
img_size = 150
```

3. Develop the Image Classification Models

Baseline Model (from scratch using Conv2D & Dense layers)

- Utilise Convolutional Neural Network (CNN or ConvNet)

  - Import libraries and packages

```
from tensorflow import keras
```

```
# Import the Required Packages
from tensorflow.keras import layers
from tensorflow.keras import models
from tensorflow.keras import optimizers
```

- Build the model

```
# Build the Model

# image size
img_size = 150

model_1A = models.Sequential()
# feature extraction layers
model_1A.add(layers.Conv2D(8, (3, 3), activation='relu',
                           input_shape=(img_size, img_size, 3)))
model_1A.add(layers.MaxPooling2D((2, 2)))
model_1A.add(layers.Conv2D(8, (3, 3), activation='relu'))
model_1A.add(layers.MaxPooling2D((2, 2)))
# classifier layers
model_1A.add(layers.Flatten())
model_1A.add(layers.Dense(4, activation='relu'))
model_1A.add(layers.Dense(10, activation='softmax'))
```

Sequential Model

Conv2D (ConvNet)

8 filters, filter size: 3x3px

Input image size: 150x150px;
Channels: 3

Max-Pool: 2x2px

To convert 3D to “1D” tensor

for multiclass, single label classification

3. Develop the Image Classification Models

Baseline Model (from scratch using conv2D & dense layers)

- Model summary

```
model_1A.summary()
```

```
Model: "sequential_26"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_49 (Conv2D)          (None, 148, 148, 8)       224

 max_pooling2d_49 (MaxPoolin  (None, 74, 74, 8)        0
 g2D)

 conv2d_50 (Conv2D)          (None, 72, 72, 8)         584

 max_pooling2d_50 (MaxPoolin  (None, 36, 36, 8)        0
 g2D)

 flatten_26 (Flatten)        (None, 10368)             0

 dense_54 (Dense)            (None, 4)                 41476

 dense_55 (Dense)            (None, 10)                50

=================================================================
Total params: 42,334
Trainable params: 42,334
Non-trainable params: 0
_________________________________________________________________
```

Feature extraction layers

Classifier layers

# of Parameters

3. Develop the Image Classification Models

Baseline Model (from scratch using Conv2D & Dense layers)

- Compile the model

Root Mean Square Propagation optimizer

Learning rate of optimizer

```
# compile the model
model_1A.compile(optimizer=optimizers.RMSprop(learning_rate=1e-3),
                 loss='categorical_crossentropy',
                 metrics=['accuracy'])
```

for multiclass, single label classification

Measure of success

- Fit (or Train) the model

```
# fit the model
history_1A = model_1A.fit(train_generator,
                          steps_per_epoch=100,  #train_sample_size/data_batch_size
                          epochs=30,
                          validation_data=validation_generator,
                          validation_steps=50)  #validation_sample_size/data_batch_size
```

```
Epoch 1/30
100/100 [==============================] - 58s 569ms/step - loss: 2.2725 - accuracy: 0.1408 - val_loss: 2.2536 - val_accuracy: 0.1460
Epoch 2/30
100/100 [==============================] - 56s 564ms/step - loss: 2.2126 - accuracy: 0.1708 - val_loss: 2.2104 - val_accuracy: 0.1635
Epoch 3/30
100/100 [==============================] - 70s 697ms/step - loss: 2.1700 - accuracy: 0.2013 - val_loss: 2.2485 - val_accuracy: 0.1775
```

3. Develop the Image Classification Models

Baseline Model (from scratch using Conv2D & Dense layers)

- Import package for plotting

```
import matplotlib.pyplot as plt
%matplotlib inline
```

- Plot the Training and Validation Accuracy & Loss Scores

```
# Plot the Training and Validation Accuracy & Loss Scores
acc_1A = history_1A.history['accuracy']
val_acc_1A = history_1A.history['val_accuracy']
loss_1A = history_1A.history['loss']
val_loss_1A = history_1A.history['val_loss']

epochs_1A = range(len(acc_1A))

plt.plot(epochs_1A, acc_1A, 'bo', label='Training acc')
plt.plot(epochs_1A, val_acc_1A, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()

plt.plot(epochs_1A, loss_1A, 'bo', label='Training loss')
plt.plot(epochs_1A, val_loss_1A, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```

Plot Accuracy to Epoch

Plot Loss to Epoch

- Save the model

```
# Save the Model
model_1A.save('/content/drive/My Drive/NP DL/ASG1/food_model_1A.h5')
```

3. Develop the Image Classification Models

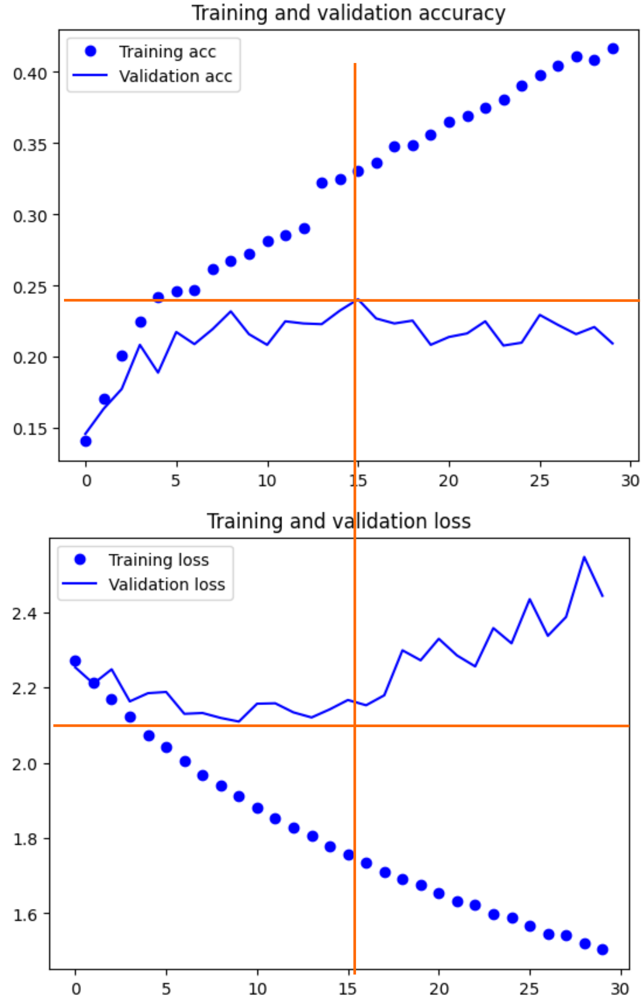
Comparison of models (regularization and hyperparameters tuning)

	model_1A	model_1B	model_1C	model_1C_2	model_1D	model_1D_2	model_1E	model_1F
Features	Baseline	Increased network size (layers, filters, nodes)	L2 weight regularization	L2 weight regularization with avg-pooling and Adam optimizer	Dropout layer	Dropout layer with avg-pooling and SGD optimizer	Reduced learning rate	Data Augmentation
No. of layers	4	6	6	6	6	6	6	6
L2 weight regularization	-	-	0.001	0.001	0.001	0.001	0.001	0.001
Type of pooling	Max-Pooling	Max-Pooling	Max-Pooling	Avg-Pooling	Max-Pooling	Avg-Pooling	Max-Pooling	Max-Pooling
Dropout layer	-	-	-	0.5	0.5	0.5	0.5	0.5
Optimizer	RMSProp(0.001)	RMSProp(0.001)	RMSProp(0.001)	Adam(0.001)	RMSProp(0.001)	SGD(0.001)	RMSProp(0.0001)	RMSProp(0.001)
Total params	42.33K	1.849M	1.849M	1.849M	1.849M	1.849M	1.849M	1.849M

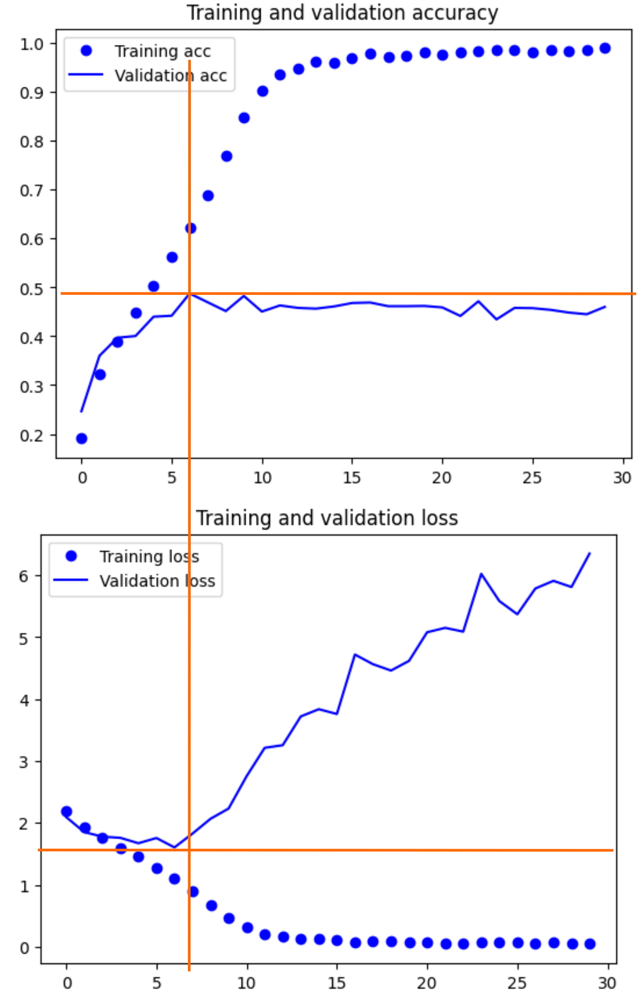
3. DEVELOP THE IMAGE CLASSIFICATION MODELS

COMPARISON OF MODELS (REGULARIZATION AND HYPERPARAMETERS TUNING)

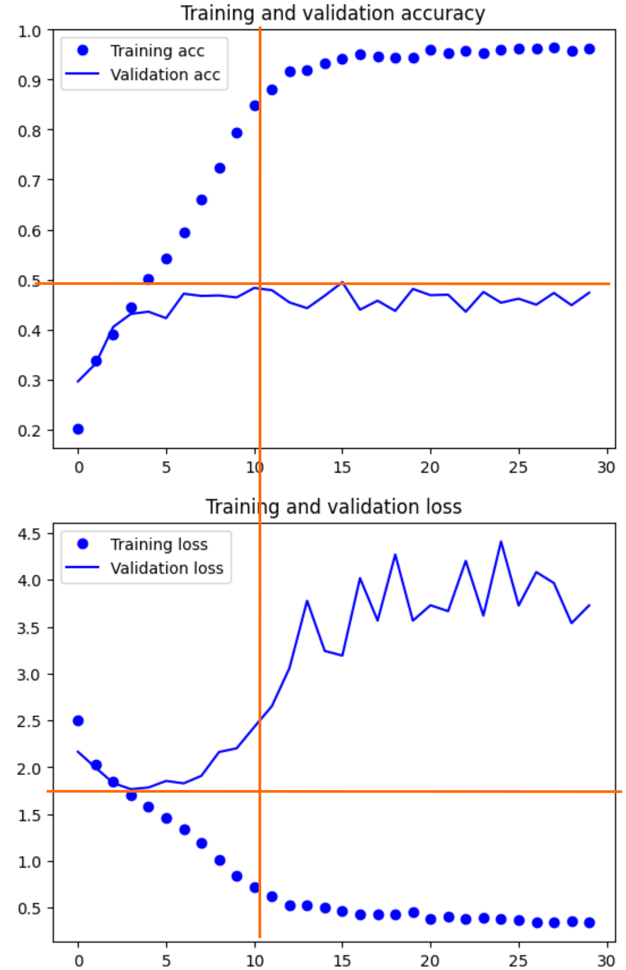
model_1A



model_1B



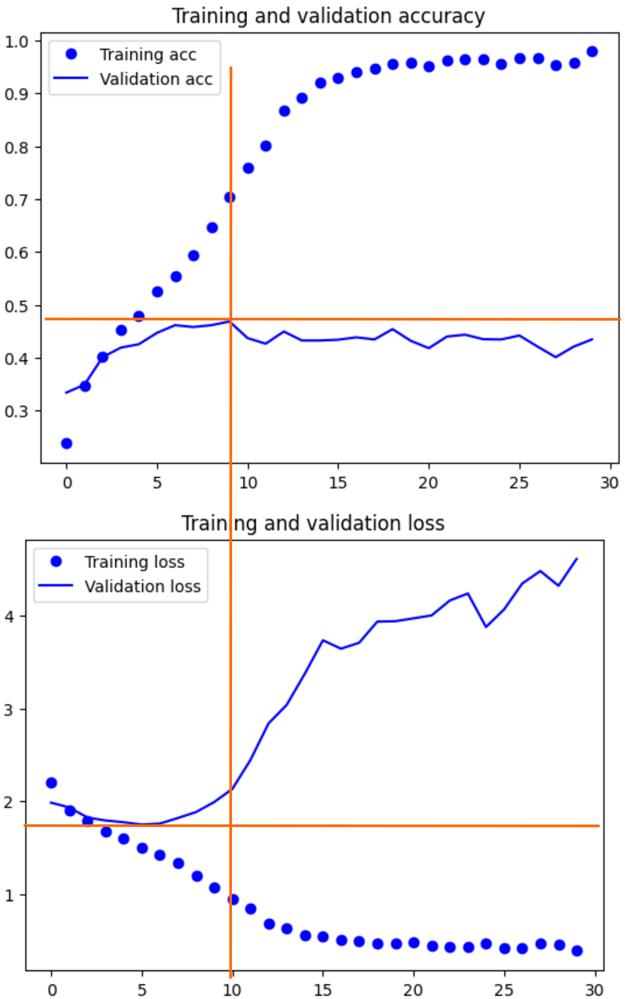
model_1C



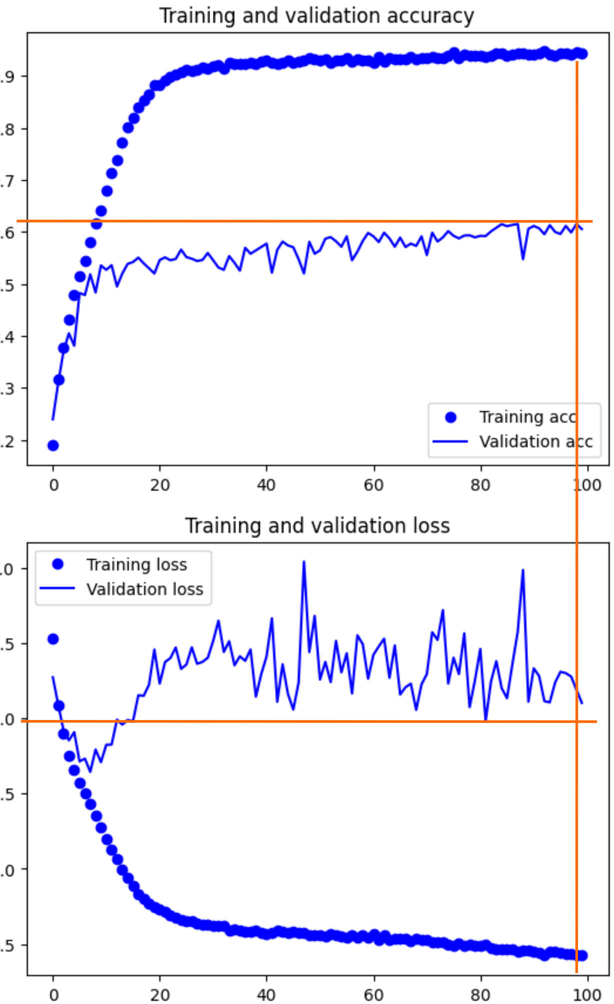
3. Develop the Image Classification Models

Comparison of models (regularization and hyperparameters tuning)

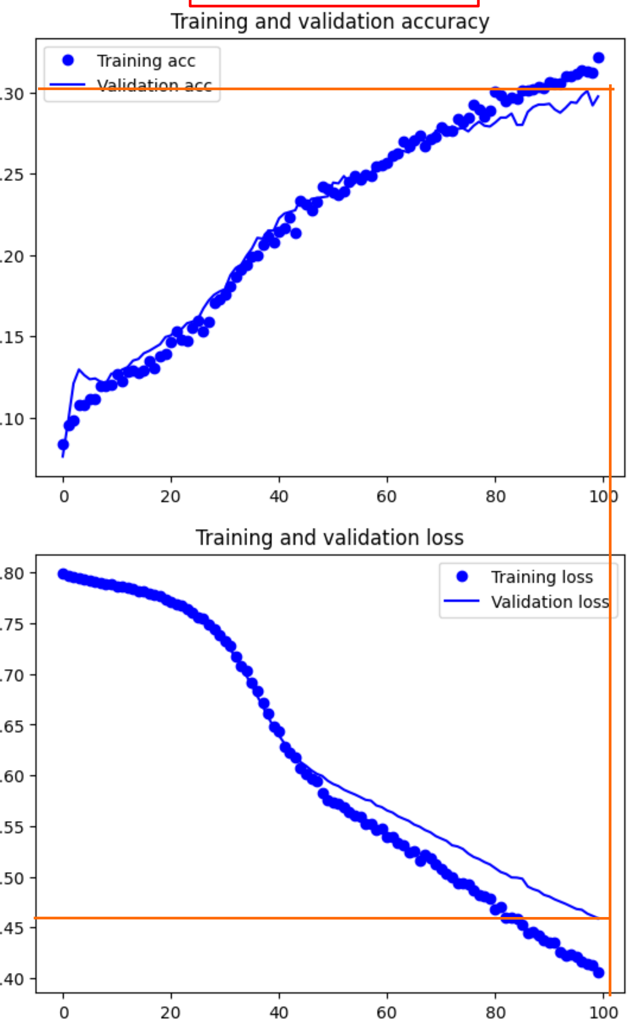
model_1C_2



model_1D

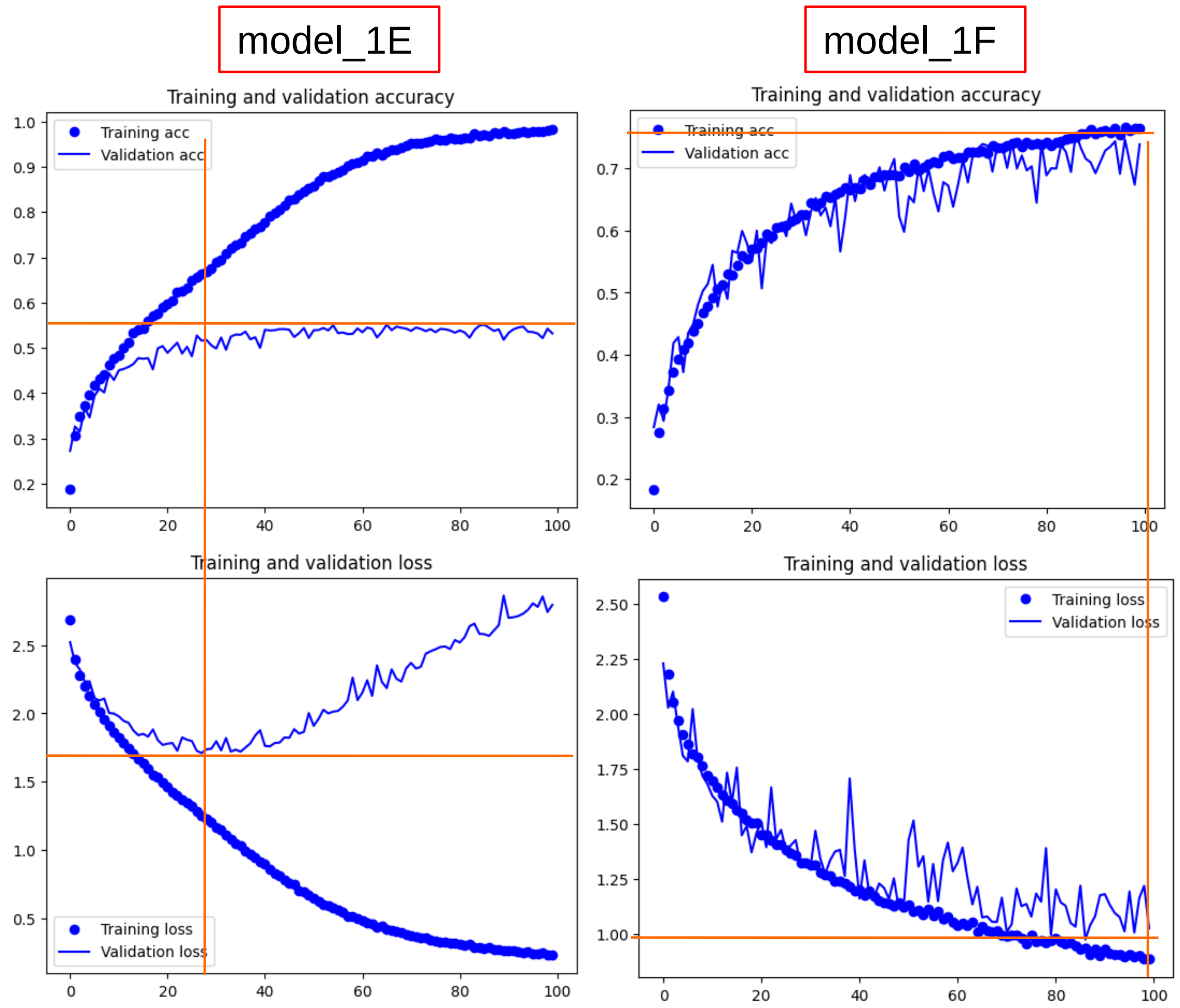


model_1D_2



3. DEVELOP THE IMAGE CLASSIFICATION MODELS

COMPARISON OF MODELS (REGULARIZATION AND HYPERPARAMETERS TUNING)



3. Develop the Image Classification Models

Comparison of models (regularization and hyperparameters tuning)

	Best Validation Accuracy (%)	Best Validation Loss	Approx. Epoch where overfitting occurs	Remarks
model_1A	24	2.1	15	-
model_1B	49	1.6	6	-
model_1C	50	1.7	10	-
model_1C_2	47	1.7	9	-
model_1D	61	2.0	-	Has not overfitted at 100th epoch. Accuracy is still increasing.
model_1D_2	30	2.46	-	Has not overfitted at 100th epoch. Accuracy is still increasing.
model_1E	55	1.7	27	-
model_1F	76	1.0	-	Has not overfitted at 100th epoch. Stagnant from around 90^{th} epoch.

3. Develop the Image Classification Models

Pretrained Model VGG16

- Utilise Pretrained Model (VGG16)

  - Import libraries and packages

```
# for pre-trained models
from tensorflow.keras.applications import VGG16
```

- Build the model

```
conv_base = VGG16(weights='imagenet',  # already pre-trained with 1000 classes with 1.4M images sample
                  include_top=False,  # exclude classifier portion
                  input_shape=(img_size, img_size, 3))

model_2B = models.Sequential()
model_2B.add(conv_base)
model_2B.add(layers.Flatten())  # with this flatten layer, we dont have to do reshaping
model_2B.add(layers.Dense(256, activation='relu'))
model_2B.add(layers.Dense(10, activation='softmax'))
```

3. Develop the Image Classification Models

Pretrained Model VGG16

- Model summary

```
model_2B.summary()
```

```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg16 (Functional)          (None, 4, 4, 512)         14714688

 flatten_5 (Flatten)         (None, 8192)              0

 dense_12 (Dense)            (None, 256)               2097408

 dense_13 (Dense)            (None, 10)                2570

=================================================================
Total params: 16,814,666
Trainable params: 2,099,978
Non-trainable params: 14,714,688
_________________________________________________________________
```

# of Parameters

3. Develop the Image Classification Models

Pretrained Model VGG16

- Compile the model

```
# compile the model
model_2B.compile(optimizer=optimizers.RMSprop(learning_rate=2e-5),
                 loss='categorical_crossentropy',
                 metrics=['accuracy'])
```

- Fit (or Train) the model

```
# fit the model
history_2B = model_2B.fit(train_generator_da_2,
                          steps_per_epoch=100,  #train_sample_size/data_batch_size
                          epochs=100,
                          validation_data=validation_generator_da_2,
                          validation_steps=50)  #validation_sample_size/data_batch_size
```

```
Epoch 1/100
100/100 [==============================] - 126s 1s/step - loss: 2.1165 - accuracy: 0.2593 - val_loss: 1.8900 - val_accuracy: 0.4010
Epoch 2/100
100/100 [==============================] - 92s 920ms/step - loss: 1.8187 - accuracy: 0.4217 - val_loss: 1.6549 - val_accuracy: 0.4890
Epoch 3/100
100/100 [==============================] - 93s 931ms/step - loss: 1.6319 - accuracy: 0.4837 - val_loss: 1.4967 - val_accuracy: 0.5320
```

3. Develop the Image Classification Models
Pretrained Model VGG16

- Plot the Training and Validation Accuracy & Loss Scores

```
# Plot the Training and Validation Accuracy & Loss Scores
acc_2B = history_2B.history['accuracy']
val_acc_2B = history_2B.history['val_accuracy']
loss_2B = history_2B.history['loss']
val_loss_2B = history_2B.history['val_loss']

epochs_2B = range(len(acc_2B))

plt.plot(epochs_2B, acc_2B, 'bo', label='Training acc')
plt.plot(epochs_2B, val_acc_2B, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()

plt.plot(epochs_2B, loss_2B, 'bo', label='Training loss')
plt.plot(epochs_2B, val_loss_2B, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
```

- Save the model

```
# Save the Model
model_2B.save('/content/drive/My Drive/NP DL/ASG1/food_model_2B.h5')
```

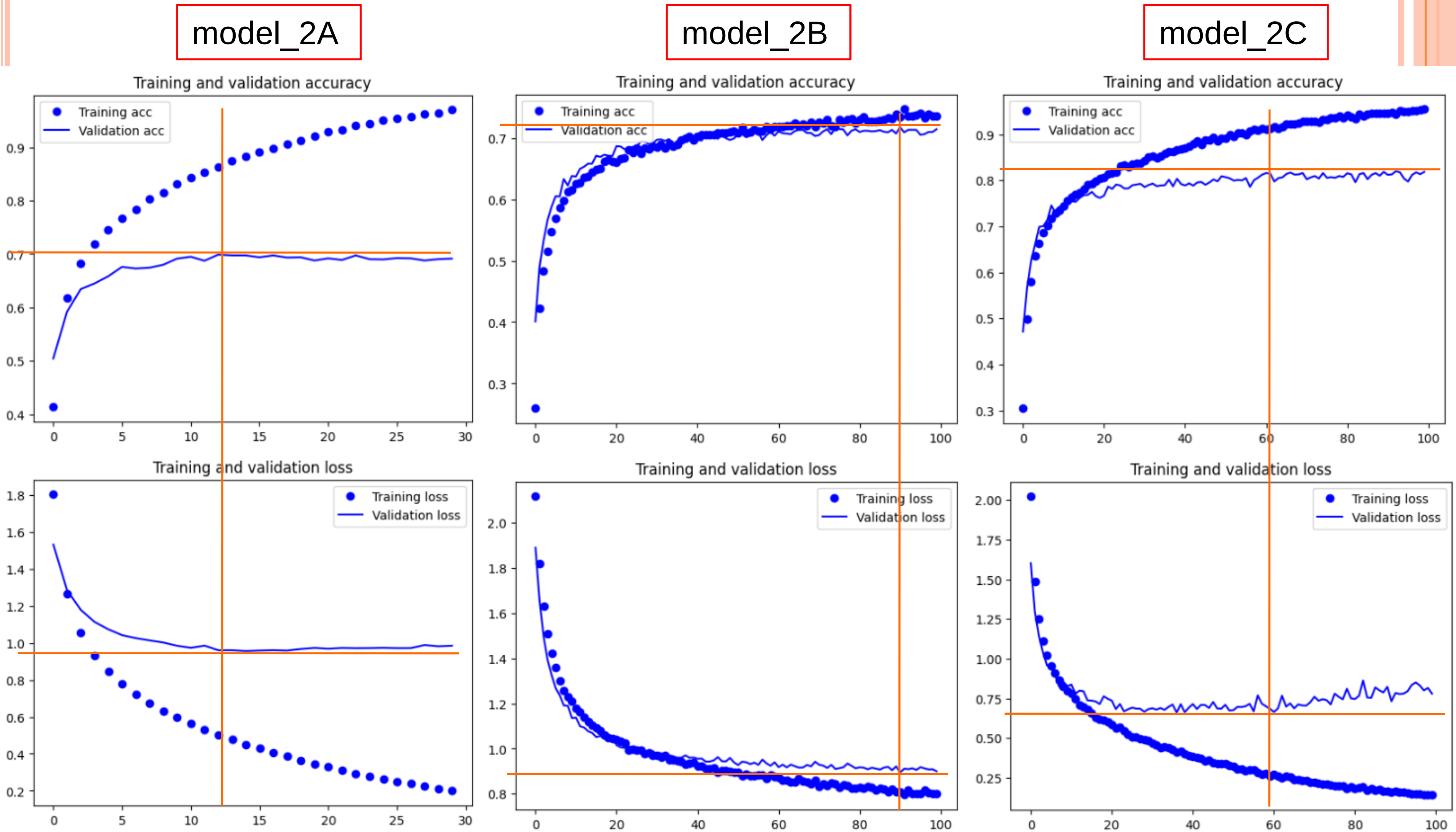
3. Develop the Image Classification Models

Comparison of models (regularization and hyperparameters tuning)

	model_2A	model_2B	model_2C	model_2D	model_2E
Features	FE w/o data augmentation	FE wtih data augmentation	FE with fine tuning and reduced learning rate (and data augmentation)	FE with dropout layer (wtih fine tuning, reduced learning rate and data augmentation)	FE with L2 weight regularization (wtih fine tuning, dropout layer, reduced learning rate and data augmentation)
Data augmentation	N	Y	Y	Y	Y
Fine tuning	-	-	Last 3 layers of conv_base	Last 3 layers of conv_base	Last 3 layers of conv_base
Optimizer learning rate	0.00002	0.00002	0.00001	0.00001	0.00001
Dropout layer	-	-	-	0.5	0.5
L2 weight regularization	-	-	-	-	0.00001
Total trainable params	14.7M	2.1M	9.2M	9.2M	9.2M

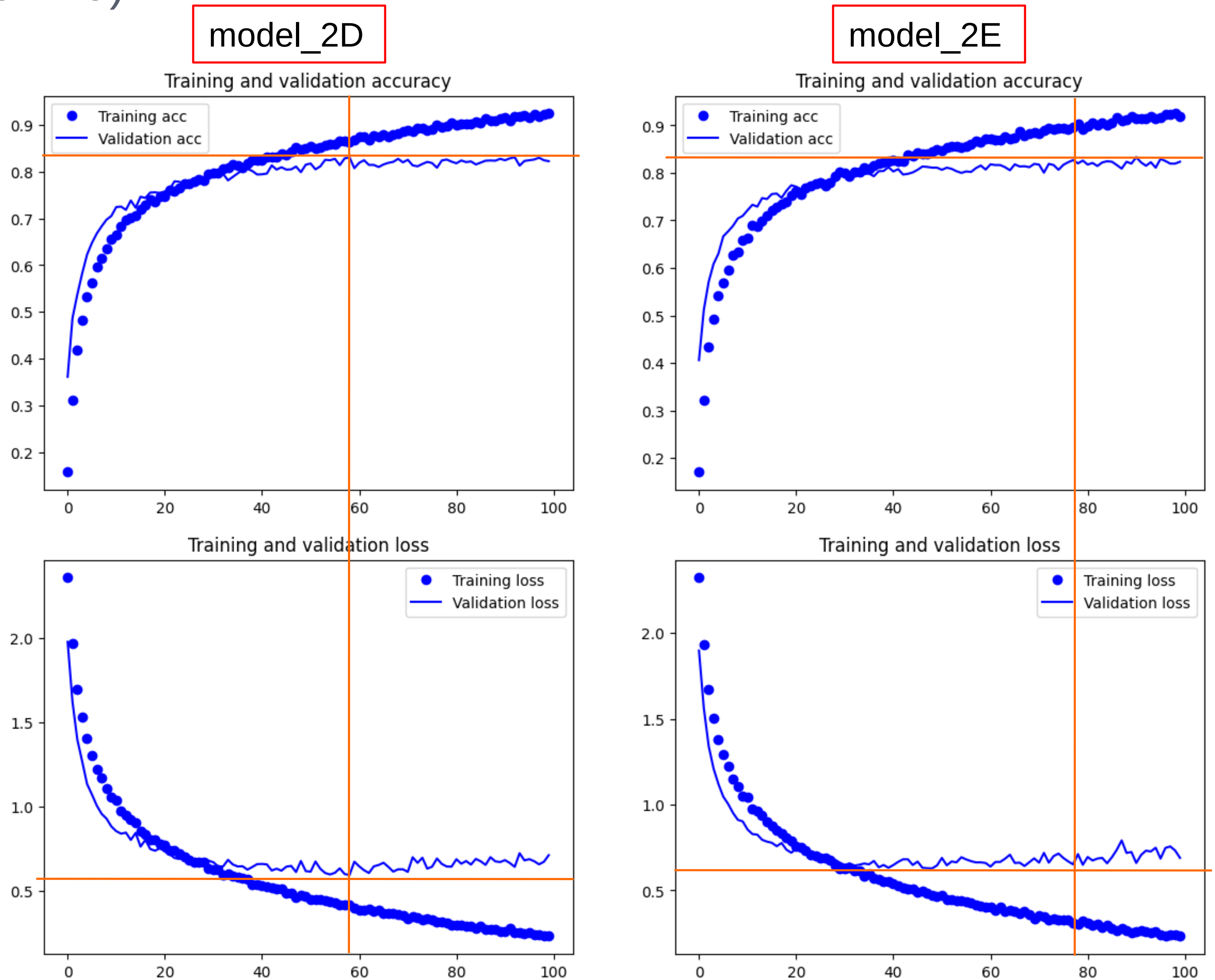
3. Develop the Image Classification Models

Comparison of models (regularization and hyperparameters tuning)



3. DEVELOP THE IMAGE CLASSIFICATION MODELS

COMPARISON OF MODELS (REGULARIZATION AND HYPERPARAMETERS TUNING)



3. DEVELOP THE IMAGE CLASSIFICATION MODELS

COMPARISON OF MODELS (REGULARIZATION AND HYPERPARAMETERS TUNING)

	Validation Accuracy	Validation Loss	Epoch where overfitting occurs	Remarks
model_2A	70	0.95	12	Overfitting may not have occurred yet. Training and validation accuracy stagnant from 12th epoch
model_2B	72	0.9	90	Overfitting may not have occurred yet. Training and validation accuracy stagnant from 90th epoch
model_2C	82	0.65	60	Overfitting may not have occurred yet. Training and validation accuracy stagnant from 60th epoch
model_2D	84	0.6	57	Overfitting may not have occurred yet. Training and validation accuracy stagnant from 58th epoch
model_2E	82	0.6	77	Overfitting may not have occurred yet. Training and validation accuracy stagnant from 77th epoch

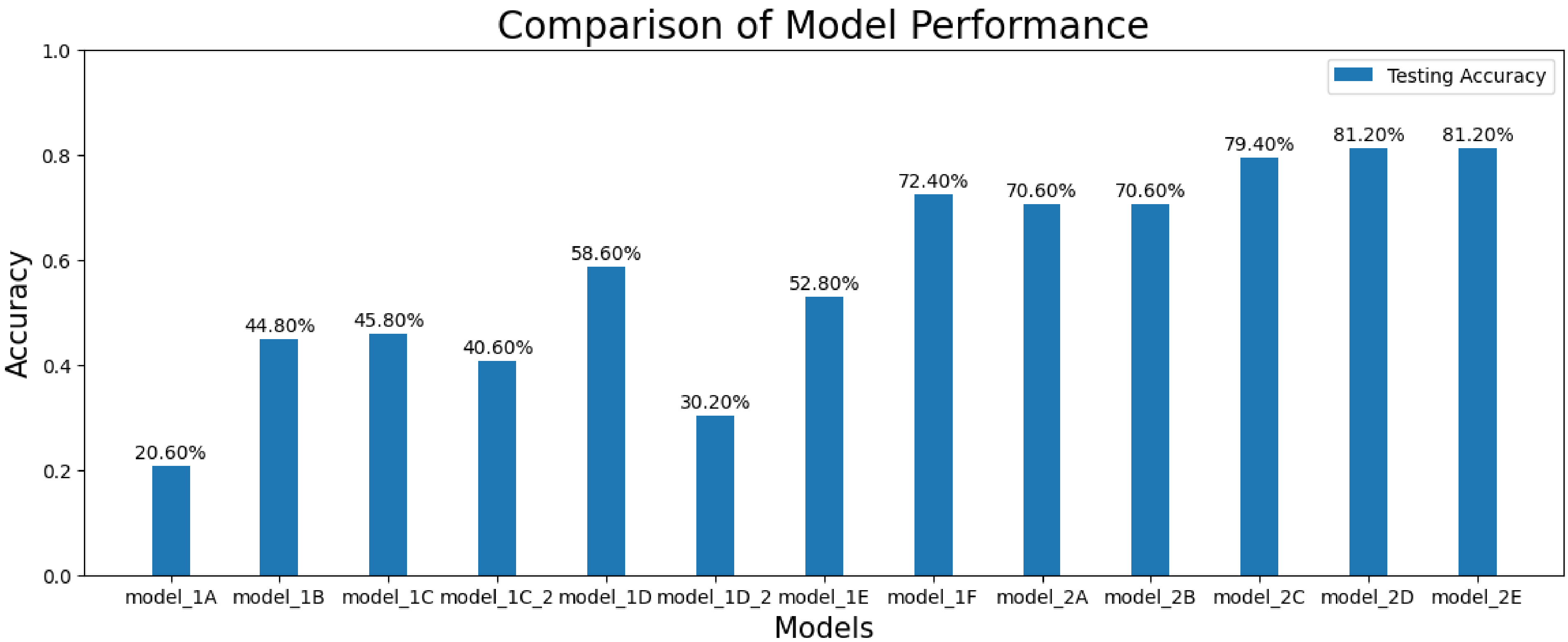
4. Evaluate the Models using Test Images

```
# Model #1A
model_1A = keras.models.load_model('/content/drive/My Drive/NP DL/ASG1/food_model_1A.h5')

# test_datagen = ImageDataGenerator(rescale=1./255)

test_generator_1A = test_datagen.flow_from_directory(
        test_dir,
        target_size=(img_size, img_size),
        batch_size=20,
        class_mode='categorical')

test_loss_1A, test_acc_1A = model_1A.evaluate(test_generator_1A, steps=25)
print('test acc:', test_acc_1A)
```

```
Found 500 images belonging to 10 classes.
25/25 [==============================] - 82s 3s/step - loss: 2.5115 - accuracy: 0.2060
test acc: 0.20600000023841858
```

Test accuracy

4. Evaluate the Models using Test Images

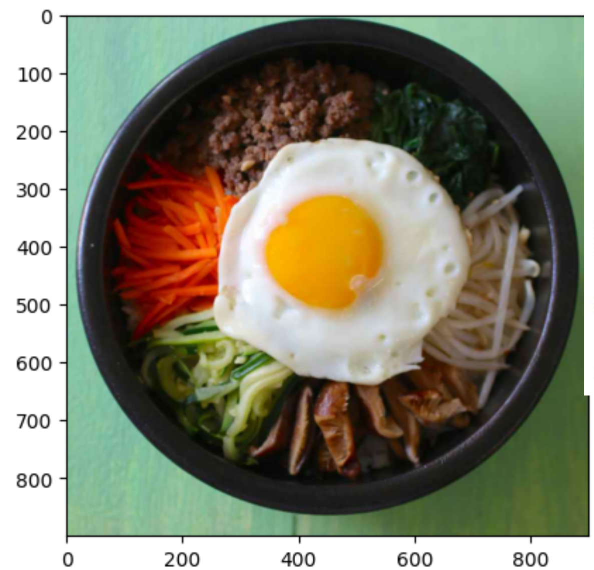


- model_2D and model_2E yielded the best performance of 81.2%
- However, according to the plotted accuracy and loss vs epoch curves, model_2E does not seem to have any better performance. This may be due to the regularization did not take effect.
- Hence, best model among these is model_2D

5. Use the Best Model to Perform Classification

- The best model, model_2D, managed to correctly classify all 3 images downloaded from the internet!


```
1/1 [==============================] - 0s 45ms/step
The prediction is  cup_cakes

   baby_back_ribs      bibimbap  cup_cakes     dumplings  fried_cal
0    4.961916e-14  6.683479e-16        1.0  3.325166e-14   1.180574

    garlic_bread       lasagna      pancakes     prime_rib      tiram
0   1.656197e-16  9.504291e-15  7.058508e-14  1.328055e-13  2.602538e
```




```
1/1 [==============================] - 1s 825ms/step
The prediction is:  bibimbap

   baby_back_ribs  bibimbap     cup_cakes     dumplings  fried_calamari
0    6.801139e-14       1.0  3.932538e-16  2.712961e-12    7.979138e-14

   garlic_bread       lasagna      pancakes     prime_rib      tiramisu
0  7.453599e-16  1.641752e-14  2.995227e-13  8.015996e-17  9.951029e-15
```


```
1/1 [==============================] - 0s 36ms/step
The prediction is:  tiramisu

   baby_back_ribs      bibimbap     cup_cakes     dumplings  fried_calamari  \
0    2.184016e-13  7.750310e-16  3.553511e-12  3.642837e-15    1.034025e-14

   garlic_bread       lasagna      pancakes     prime_rib  tiramisu
0  2.763647e-13  1.239966e-11  3.667415e-13  4.644287e-13       1.0
```

6. CONCLUSION

- Improvements
  - Use other pretrained models
  - Continue to tune the hyperparameters to get even better performance
  - Continue to fine-tune the conv_base layers