# transferlearningassignment

June 13, 2023

## 1 Computer Vision Assignment 04


#### 1.0.1 Name: Ahad Anwar

#### 1.0.2 Reg: SP20-BCS-054


---

## 2 VGG19

### 2.1 Applying VGG19 for Transfer Learning

[1]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

[2]:
```
import os
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import VGG19
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D
from tensorflow.keras.models import Model
from sklearn.model_selection import train_test_split
```

[3]:
```
base_path = '/content/drive/MyDrive/dataset/images'
pizza_folder = os.path.join(base_path, 'pizza')
burger_folder = os.path.join(base_path, 'burger')
icecream_folder = os.path.join(base_path, 'icecream')
```

[4]:
```
image_size = (224, 224)
batch_size = 32

datagen = ImageDataGenerator(
    rescale=1./255,
    validation_split=0.2,  # 80% training, 20% validation split
    rotation_range=20,
```

```
    width_shift_range=0.2,
    height_shift_range=0.2,
    horizontal_flip=True
)
```

[16]:
```
train_generator = datagen.flow_from_directory(
    base_path,
    target_size=image_size,
    batch_size=batch_size,
    class_mode='categorical',
    subset='training'
)

validation_generator = datagen.flow_from_directory(
    base_path,
    target_size=image_size,
    batch_size=batch_size,
    class_mode='categorical',
    subset='validation'
)
```

```
Found 243 images belonging to 3 classes.
Found 60 images belonging to 3 classes.
```

[6]:
```
base_model = VGG19(weights='imagenet', include_top=False, input_shape=(224,␣
 ↪224, 3))

for layer in base_model.layers:
    layer.trainable = False
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/vgg19/vgg19_weights_tf_dim_ordering_tf_kernels_notop.h5
80134624/80134624 [==============================] - 3s 0us/step
```

[7]:
```
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(256, activation='relu')(x)
predictions = Dense(3, activation='softmax')(x)

model = Model(inputs=base_model.input, outputs=predictions)
```

[9]:
```
model.compile(optimizer='adam', loss='categorical_crossentropy',␣
 ↪metrics=['accuracy'])
model.fit(train_generator, validation_data=validation_generator, epochs=30)
```

```
Epoch 1/30
8/8 [==============================] - 17s 2s/step - loss: 0.3609 - accuracy:
```

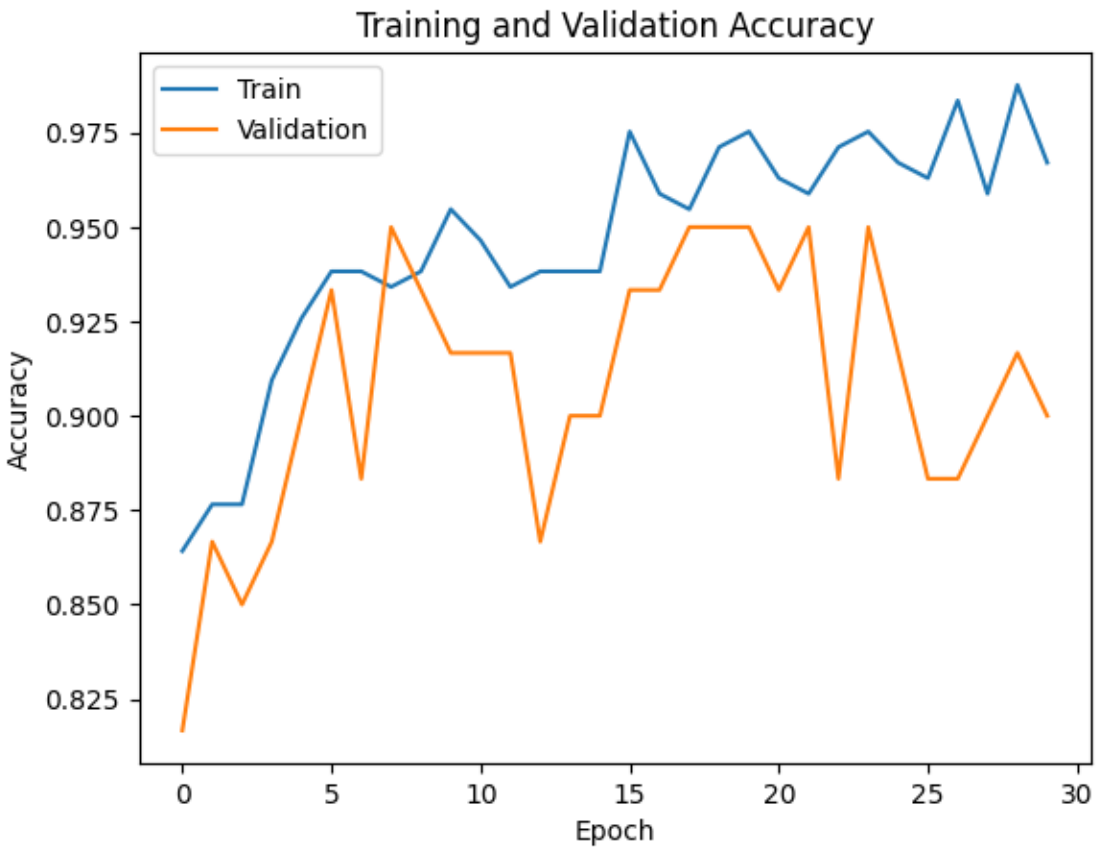
```
0.8642 - val_loss: 0.4569 - val_accuracy: 0.8167
Epoch 2/30
8/8 [==============================] - 15s 2s/step - loss: 0.3345 - accuracy:
0.8765 - val_loss: 0.3473 - val_accuracy: 0.8667
Epoch 3/30
8/8 [==============================] - 14s 2s/step - loss: 0.3039 - accuracy:
0.8765 - val_loss: 0.4192 - val_accuracy: 0.8500
Epoch 4/30
8/8 [==============================] - 15s 2s/step - loss: 0.2854 - accuracy:
0.9095 - val_loss: 0.3975 - val_accuracy: 0.8667
Epoch 5/30
8/8 [==============================] - 15s 2s/step - loss: 0.2598 - accuracy:
0.9259 - val_loss: 0.3352 - val_accuracy: 0.9000
Epoch 6/30
8/8 [==============================] - 15s 2s/step - loss: 0.2311 - accuracy:
0.9383 - val_loss: 0.3102 - val_accuracy: 0.9333
Epoch 7/30
8/8 [==============================] - 15s 2s/step - loss: 0.2202 - accuracy:
0.9383 - val_loss: 0.3131 - val_accuracy: 0.8833
Epoch 8/30
8/8 [==============================] - 14s 2s/step - loss: 0.2246 - accuracy:
0.9342 - val_loss: 0.2999 - val_accuracy: 0.9500
Epoch 9/30
8/8 [==============================] - 15s 2s/step - loss: 0.2266 - accuracy:
0.9383 - val_loss: 0.2732 - val_accuracy: 0.9333
Epoch 10/30
8/8 [==============================] - 17s 2s/step - loss: 0.1871 - accuracy:
0.9547 - val_loss: 0.3084 - val_accuracy: 0.9167
Epoch 11/30
8/8 [==============================] - 15s 2s/step - loss: 0.1789 - accuracy:
0.9465 - val_loss: 0.2663 - val_accuracy: 0.9167
Epoch 12/30
8/8 [==============================] - 15s 2s/step - loss: 0.1791 - accuracy:
0.9342 - val_loss: 0.2597 - val_accuracy: 0.9167
Epoch 13/30
8/8 [==============================] - 15s 2s/step - loss: 0.1876 - accuracy:
0.9383 - val_loss: 0.3402 - val_accuracy: 0.8667
Epoch 14/30
8/8 [==============================] - 15s 2s/step - loss: 0.1762 - accuracy:
0.9383 - val_loss: 0.2634 - val_accuracy: 0.9000
Epoch 15/30
8/8 [==============================] - 17s 2s/step - loss: 0.1753 - accuracy:
0.9383 - val_loss: 0.2973 - val_accuracy: 0.9000
Epoch 16/30
8/8 [==============================] - 15s 2s/step - loss: 0.1416 - accuracy:
0.9753 - val_loss: 0.2532 - val_accuracy: 0.9333
Epoch 17/30
8/8 [==============================] - 15s 2s/step - loss: 0.1466 - accuracy:
```

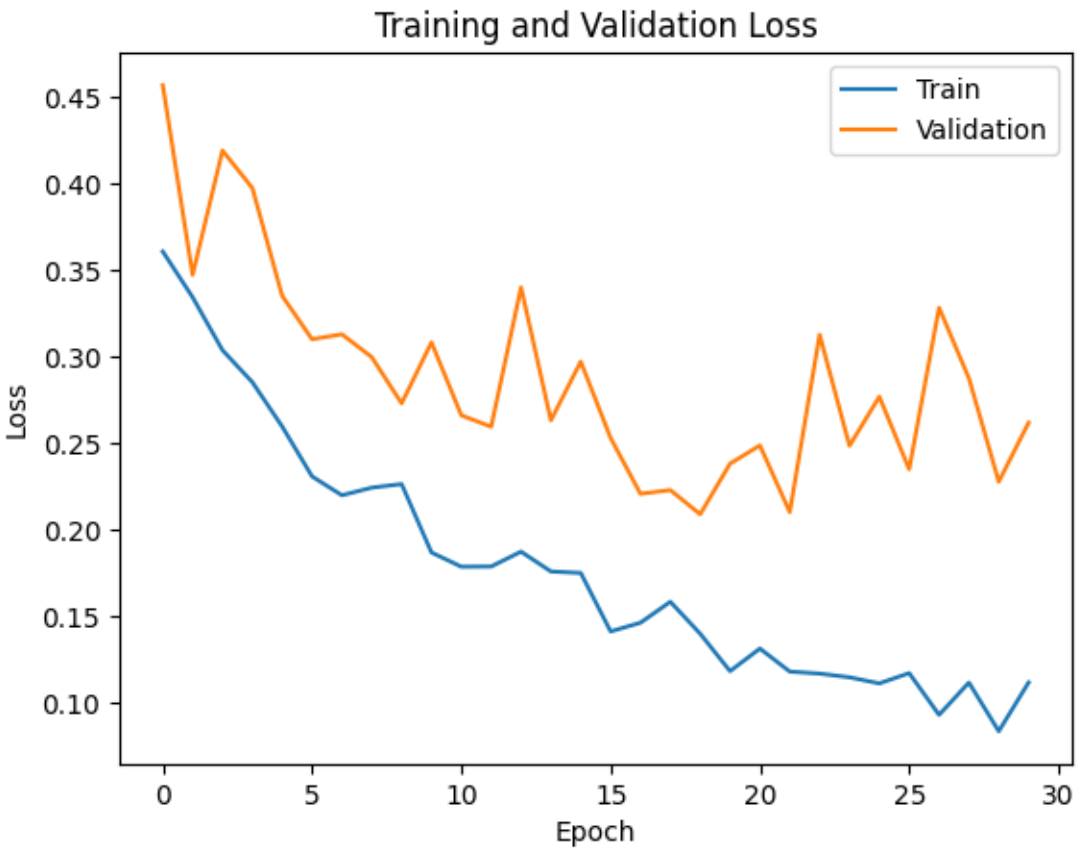
```
0.9588 - val_loss: 0.2210 - val_accuracy: 0.9333
Epoch 18/30
8/8 [==============================] - 15s 2s/step - loss: 0.1587 - accuracy:
0.9547 - val_loss: 0.2232 - val_accuracy: 0.9500
Epoch 19/30
8/8 [==============================] - 15s 2s/step - loss: 0.1403 - accuracy:
0.9712 - val_loss: 0.2092 - val_accuracy: 0.9500
Epoch 20/30
8/8 [==============================] - 17s 2s/step - loss: 0.1187 - accuracy:
0.9753 - val_loss: 0.2384 - val_accuracy: 0.9500
Epoch 21/30
8/8 [==============================] - 15s 2s/step - loss: 0.1317 - accuracy:
0.9630 - val_loss: 0.2489 - val_accuracy: 0.9333
Epoch 22/30
8/8 [==============================] - 15s 2s/step - loss: 0.1184 - accuracy:
0.9588 - val_loss: 0.2105 - val_accuracy: 0.9500
Epoch 23/30
8/8 [==============================] - 15s 2s/step - loss: 0.1172 - accuracy:
0.9712 - val_loss: 0.3129 - val_accuracy: 0.8833
Epoch 24/30
8/8 [==============================] - 15s 2s/step - loss: 0.1151 - accuracy:
0.9753 - val_loss: 0.2487 - val_accuracy: 0.9500
Epoch 25/30
8/8 [==============================] - 15s 2s/step - loss: 0.1116 - accuracy:
0.9671 - val_loss: 0.2771 - val_accuracy: 0.9167
Epoch 26/30
8/8 [==============================] - 15s 2s/step - loss: 0.1175 - accuracy:
0.9630 - val_loss: 0.2352 - val_accuracy: 0.8833
Epoch 27/30
8/8 [==============================] - 15s 2s/step - loss: 0.0934 - accuracy:
0.9835 - val_loss: 0.3284 - val_accuracy: 0.8833
Epoch 28/30
8/8 [==============================] - 15s 2s/step - loss: 0.1121 - accuracy:
0.9588 - val_loss: 0.2877 - val_accuracy: 0.9000
Epoch 29/30
8/8 [==============================] - 15s 2s/step - loss: 0.0840 - accuracy:
0.9877 - val_loss: 0.2280 - val_accuracy: 0.9167
Epoch 30/30
8/8 [==============================] - 15s 2s/step - loss: 0.1121 - accuracy:
0.9671 - val_loss: 0.2621 - val_accuracy: 0.9000
```

[9]:
```
<keras.callbacks.History at 0x7fa977c50640>
```

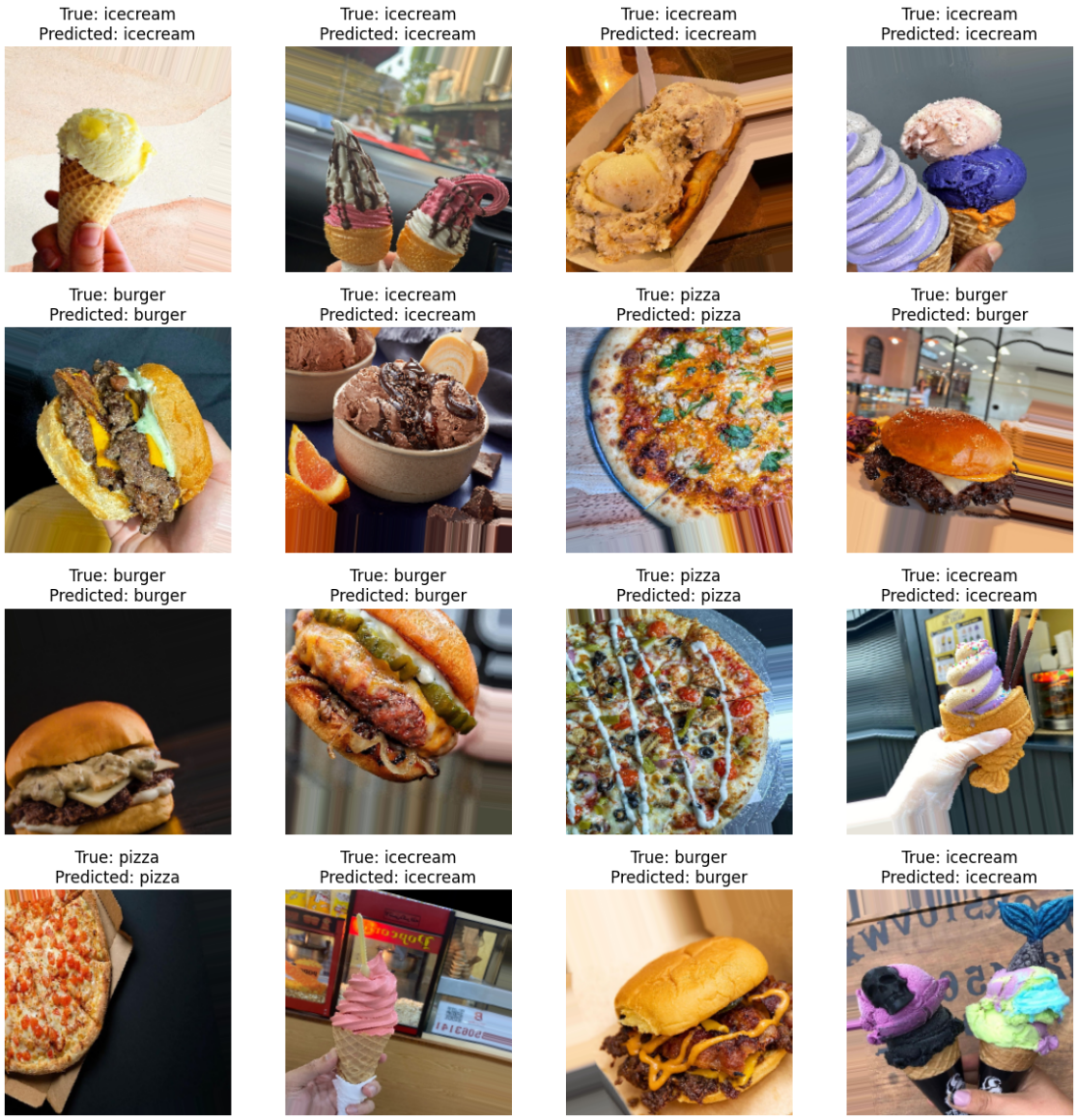
[10]:
```python
import matplotlib.pyplot as plt

# Plot training and validation accuracy
plt.plot(model.history.history['accuracy'])
```

```
plt.plot(model.history.history['val_accuracy'])
plt.title('Training and Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(['Train', 'Validation'])
plt.show()

# Plot training and validation loss
plt.plot(model.history.history['loss'])
plt.plot(model.history.history['val_loss'])
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(['Train', 'Validation'])
plt.show()

# Generate sample predictions
sample_images, sample_labels = next(validation_generator)
sample_predictions = model.predict(sample_images)

# Convert one-hot encoded labels back to class names
class_names = list(train_generator.class_indices.keys())
sample_labels = np.argmax(sample_labels, axis=1)
sample_predictions = np.argmax(sample_predictions, axis=1)

# Display sample images with their true and predicted labels
fig, axes = plt.subplots(4, 4, figsize=(12, 12))
axes = axes.flatten()

for i, ax in enumerate(axes):
    ax.imshow(sample_images[i])
    ax.set_title(f'True: {class_names[sample_labels[i]]}\nPredicted:␣
 ↪{class_names[sample_predictions[i]]}')
    ax.axis('off')

plt.tight_layout()
plt.show()
```


Training and Validation Accuracy
Train
Validation
Accuracy
0.975
0.950
0.925
0.900
0.875
0.850
0.825
0
5
10
15
20
25
30
Epoch




```
1/1 [==============================] - 0s 212ms/step
```




---

# 3 Inception V3

## 3.1 Applying Inception V3 for Transfer Learning

[25]:
```
from tensorflow.keras.applications import InceptionV3
```

[26]:
```
train_generator = datagen.flow_from_directory(
    base_path,
    target_size=image_size,
    batch_size=batch_size,
```

```
    class_mode='categorical',
    subset='training',
    shuffle=True
)

validation_generator = datagen.flow_from_directory(
    base_path,
    target_size=image_size,
    batch_size=batch_size,
    class_mode='categorical',
    subset='validation',
    shuffle=True
)
```

```
Found 243 images belonging to 3 classes.
Found 60 images belonging to 3 classes.
```

[27]:
```
base_model = InceptionV3(weights='imagenet', include_top=False,␣
 ↪input_shape=(224, 224, 3))

for layer in base_model.layers:
    layer.trainable = False
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applicatio
ns/inception_v3/inception_v3_weights_tf_dim_ordering_tf_kernels_notop.h5
87910968/87910968 [==============================] - 0s 0us/step
```

[28]:
```
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(256, activation='relu')(x)
predictions = Dense(3, activation='softmax')(x)

model = Model(inputs=base_model.input, outputs=predictions)
```

[29]:
```
model.compile(optimizer='adam', loss='categorical_crossentropy',␣
 ↪metrics=['accuracy'])
model.fit(train_generator, validation_data=validation_generator, epochs=30)
```

```
Epoch 1/30
8/8 [==============================] - 26s 3s/step - loss: 1.5165 - accuracy:
0.6502 - val_loss: 0.4087 - val_accuracy: 0.8667
Epoch 2/30
8/8 [==============================] - 15s 2s/step - loss: 0.2634 - accuracy:
0.9218 - val_loss: 0.1328 - val_accuracy: 0.9333
Epoch 3/30
8/8 [==============================] - 15s 2s/step - loss: 0.1086 - accuracy:
0.9712 - val_loss: 0.1321 - val_accuracy: 0.9500
Epoch 4/30
```

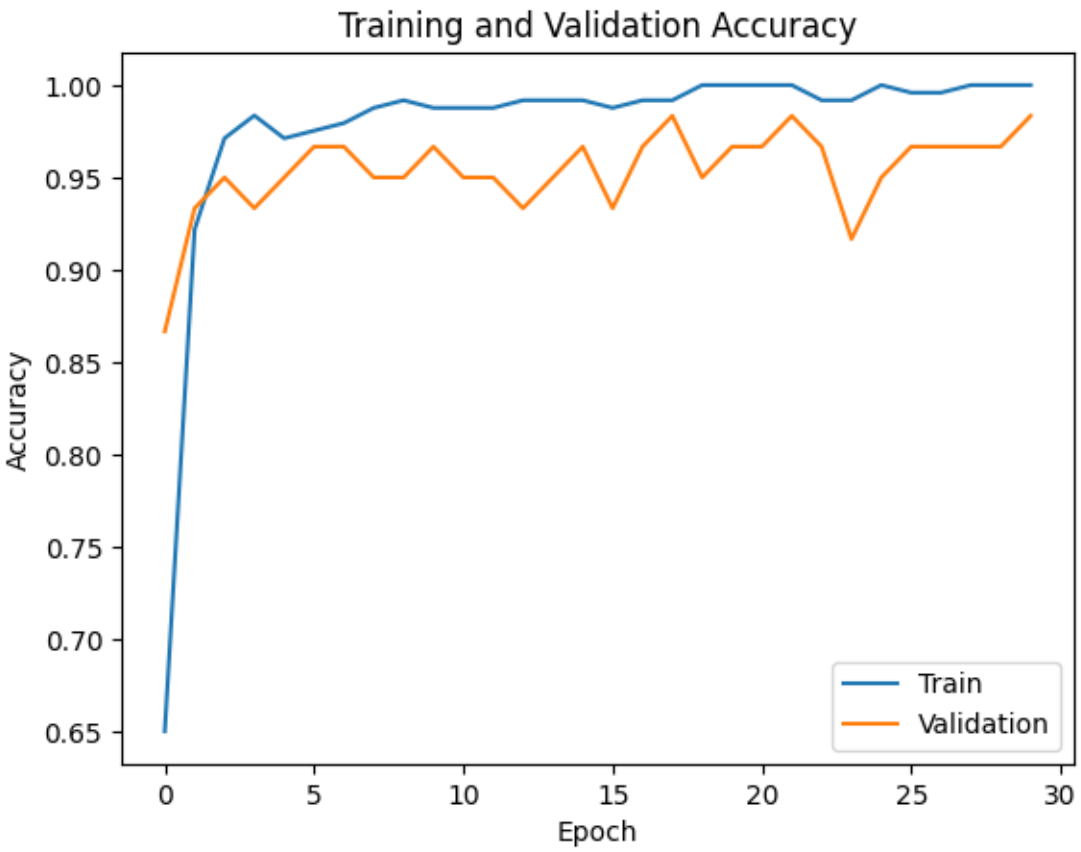
```
8/8 [==============================] - 15s 2s/step - loss: 0.0584 - accuracy:
0.9835 - val_loss: 0.1278 - val_accuracy: 0.9333
Epoch 5/30
8/8 [==============================] - 16s 2s/step - loss: 0.0758 - accuracy:
0.9712 - val_loss: 0.1229 - val_accuracy: 0.9500
Epoch 6/30
8/8 [==============================] - 17s 2s/step - loss: 0.0561 - accuracy:
0.9753 - val_loss: 0.0742 - val_accuracy: 0.9667
Epoch 7/30
8/8 [==============================] - 15s 2s/step - loss: 0.0377 - accuracy:
0.9794 - val_loss: 0.1600 - val_accuracy: 0.9667
Epoch 8/30
8/8 [==============================] - 17s 2s/step - loss: 0.0227 - accuracy:
0.9877 - val_loss: 0.1534 - val_accuracy: 0.9500
Epoch 9/30
8/8 [==============================] - 15s 2s/step - loss: 0.0259 - accuracy:
0.9918 - val_loss: 0.0573 - val_accuracy: 0.9500
Epoch 10/30
8/8 [==============================] - 15s 2s/step - loss: 0.0265 - accuracy:
0.9877 - val_loss: 0.1034 - val_accuracy: 0.9667
Epoch 11/30
8/8 [==============================] - 15s 2s/step - loss: 0.0385 - accuracy:
0.9877 - val_loss: 0.1334 - val_accuracy: 0.9500
Epoch 12/30
8/8 [==============================] - 15s 2s/step - loss: 0.0362 - accuracy:
0.9877 - val_loss: 0.0930 - val_accuracy: 0.9500
Epoch 13/30
8/8 [==============================] - 15s 2s/step - loss: 0.0133 - accuracy:
0.9918 - val_loss: 0.1483 - val_accuracy: 0.9333
Epoch 14/30
8/8 [==============================] - 15s 2s/step - loss: 0.0129 - accuracy:
0.9918 - val_loss: 0.2800 - val_accuracy: 0.9500
Epoch 15/30
8/8 [==============================] - 15s 2s/step - loss: 0.0264 - accuracy:
0.9918 - val_loss: 0.0816 - val_accuracy: 0.9667
Epoch 16/30
8/8 [==============================] - 15s 2s/step - loss: 0.0490 - accuracy:
0.9877 - val_loss: 0.2203 - val_accuracy: 0.9333
Epoch 17/30
8/8 [==============================] - 15s 2s/step - loss: 0.0131 - accuracy:
0.9918 - val_loss: 0.1000 - val_accuracy: 0.9667
Epoch 18/30
8/8 [==============================] - 17s 2s/step - loss: 0.0142 - accuracy:
0.9918 - val_loss: 0.0737 - val_accuracy: 0.9833
Epoch 19/30
8/8 [==============================] - 15s 2s/step - loss: 0.0078 - accuracy:
1.0000 - val_loss: 0.1870 - val_accuracy: 0.9500
Epoch 20/30
```

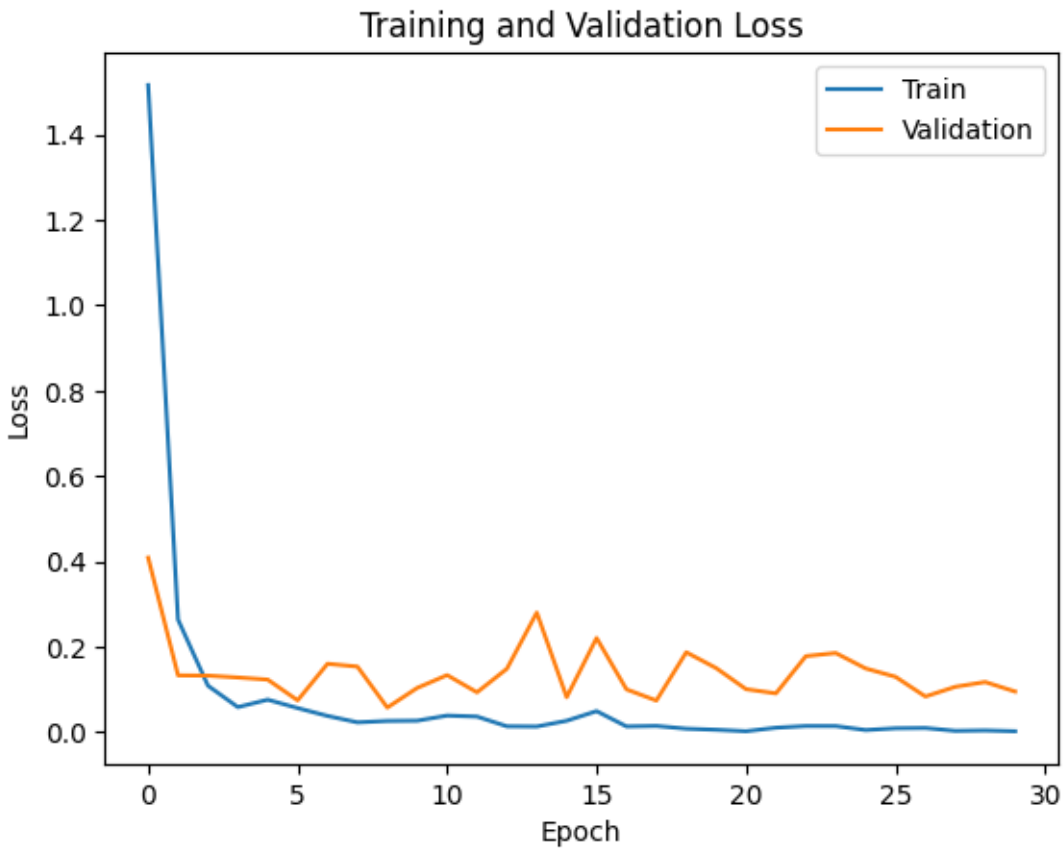
```
8/8 [==============================] - 15s 2s/step - loss: 0.0053 - accuracy:
1.0000 - val_loss: 0.1499 - val_accuracy: 0.9667
Epoch 21/30
8/8 [==============================] - 17s 2s/step - loss: 0.0019 - accuracy:
1.0000 - val_loss: 0.1004 - val_accuracy: 0.9667
Epoch 22/30
8/8 [==============================] - 15s 2s/step - loss: 0.0101 - accuracy:
1.0000 - val_loss: 0.0906 - val_accuracy: 0.9833
Epoch 23/30
8/8 [==============================] - 15s 2s/step - loss: 0.0140 - accuracy:
0.9918 - val_loss: 0.1779 - val_accuracy: 0.9667
Epoch 24/30
8/8 [==============================] - 15s 2s/step - loss: 0.0139 - accuracy:
0.9918 - val_loss: 0.1852 - val_accuracy: 0.9167
Epoch 25/30
8/8 [==============================] - 15s 2s/step - loss: 0.0046 - accuracy:
1.0000 - val_loss: 0.1490 - val_accuracy: 0.9500
Epoch 26/30
8/8 [==============================] - 15s 2s/step - loss: 0.0087 - accuracy:
0.9959 - val_loss: 0.1294 - val_accuracy: 0.9667
Epoch 27/30
8/8 [==============================] - 15s 2s/step - loss: 0.0094 - accuracy:
0.9959 - val_loss: 0.0832 - val_accuracy: 0.9667
Epoch 28/30
8/8 [==============================] - 15s 2s/step - loss: 0.0028 - accuracy:
1.0000 - val_loss: 0.1060 - val_accuracy: 0.9667
Epoch 29/30
8/8 [==============================] - 17s 2s/step - loss: 0.0037 - accuracy:
1.0000 - val_loss: 0.1172 - val_accuracy: 0.9667
Epoch 30/30
8/8 [==============================] - 17s 2s/step - loss: 0.0019 - accuracy:
1.0000 - val_loss: 0.0949 - val_accuracy: 0.9833
```

[29]:
```
<keras.callbacks.History at 0x7f9b3eee3e50>
```

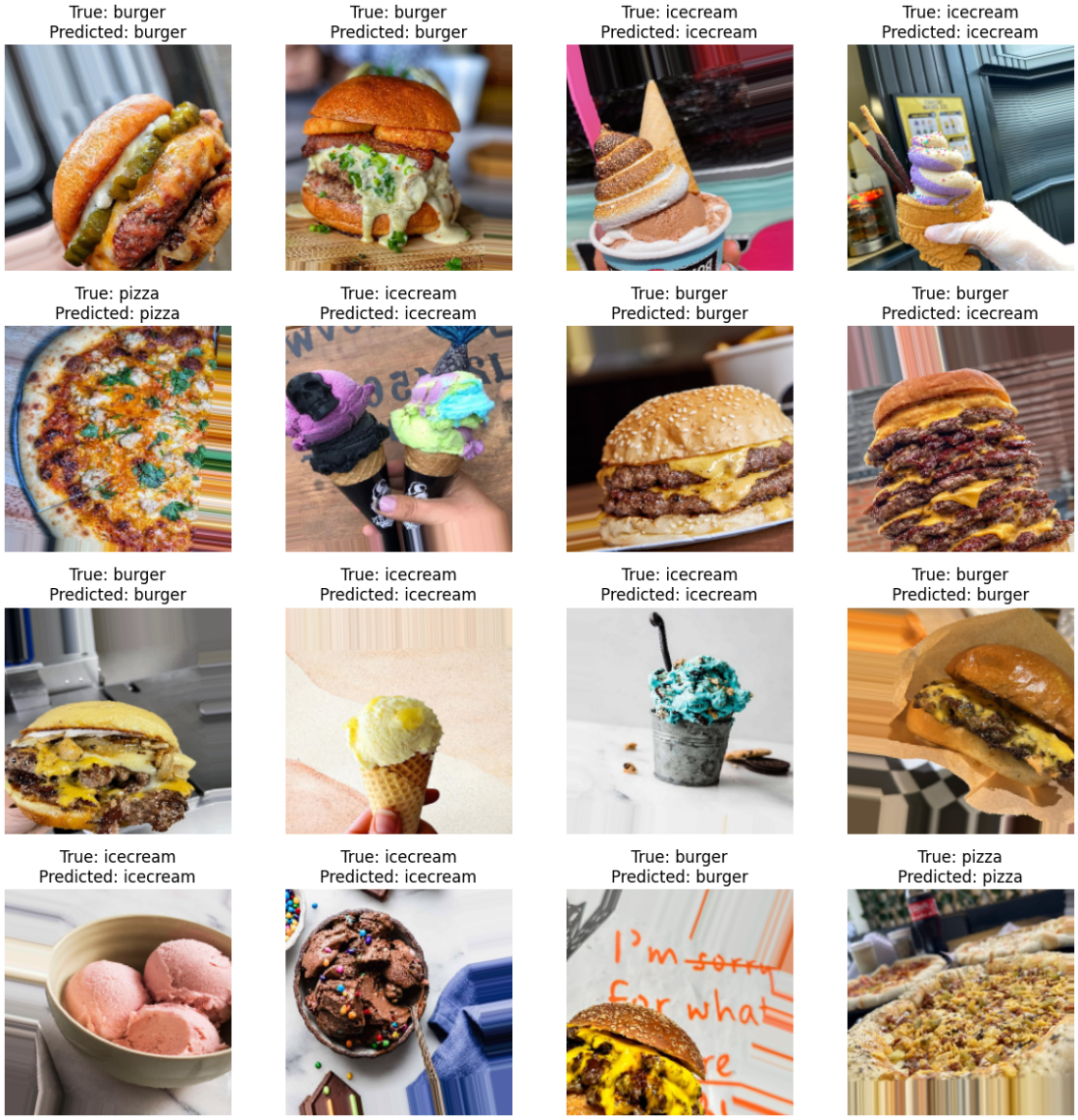
[30]:
```python
import matplotlib.pyplot as plt

# Plot training and validation accuracy
plt.plot(model.history.history['accuracy'])
plt.plot(model.history.history['val_accuracy'])
plt.title('Training and Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(['Train', 'Validation'])
plt.show()

# Plot training and validation loss
```

```
plt.plot(model.history.history['loss'])
plt.plot(model.history.history['val_loss'])
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(['Train', 'Validation'])
plt.show()

# Generate sample predictions
sample_images, sample_labels = next(validation_generator)
sample_predictions = model.predict(sample_images)

# Convert one-hot encoded labels back to class names
class_names = list(train_generator.class_indices.keys())
sample_labels = np.argmax(sample_labels, axis=1)
sample_predictions = np.argmax(sample_predictions, axis=1)

# Display sample images with their true and predicted labels
fig, axes = plt.subplots(4, 4, figsize=(12, 12))
axes = axes.flatten()

for i, ax in enumerate(axes):
    ax.imshow(sample_images[i])
    ax.set_title(f'True: {class_names[sample_labels[i]]}\nPredicted:␣
 ↪{class_names[sample_predictions[i]]}')
    ax.axis('off')

plt.tight_layout()
plt.show()
```


Training and Validation Accuracy
Accuracy
Epoch
Train
Validation
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0
5
10
15
20
25
30


Training and Validation Loss
Train
Validation
Loss
Epoch
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
0
5
10
15
20
25
30

```
1/1 [==============================] - 1s 1s/step
```




---

# 4 Resnet50

## 4.1 Applying Resnet50 for Transfer Learning

[26]:
```
from tensorflow.keras.applications import ResNet50
```

[27]:
```
train_generator = datagen.flow_from_directory(
    base_path,
    target_size=image_size,
    batch_size=batch_size,
```

```
    class_mode='categorical',
    subset='training',
    shuffle=True
)

validation_generator = datagen.flow_from_directory(
    base_path,
    target_size=image_size,
    batch_size=batch_size,
    class_mode='categorical',
    subset='validation',
    shuffle=True
)
```

```
Found 243 images belonging to 3 classes.
Found 60 images belonging to 3 classes.
```

[28]:
```
base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(224,␣
↪224, 3))

for layer in base_model.layers:
    layer.trainable = False
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5
94765736/94765736 [==============================] - 1s 0us/step
```

[29]:
```
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(256, activation='relu')(x)
predictions = Dense(3, activation='softmax')(x)

model = Model(inputs=base_model.input, outputs=predictions)
```

[30]:
```
model.compile(optimizer='adam', loss='categorical_crossentropy',␣
↪metrics=['accuracy'])
model.fit(train_generator, validation_data=validation_generator, epochs=30)
```

```
Epoch 1/30
8/8 [==============================] - 21s 2s/step - loss: 1.3508 - accuracy:
0.3580 - val_loss: 1.1428 - val_accuracy: 0.3667
Epoch 2/30
8/8 [==============================] - 14s 2s/step - loss: 1.1880 - accuracy:
0.3663 - val_loss: 1.1953 - val_accuracy: 0.4333
Epoch 3/30
8/8 [==============================] - 14s 2s/step - loss: 1.1340 - accuracy:
0.4321 - val_loss: 0.9911 - val_accuracy: 0.5667
Epoch 4/30
```

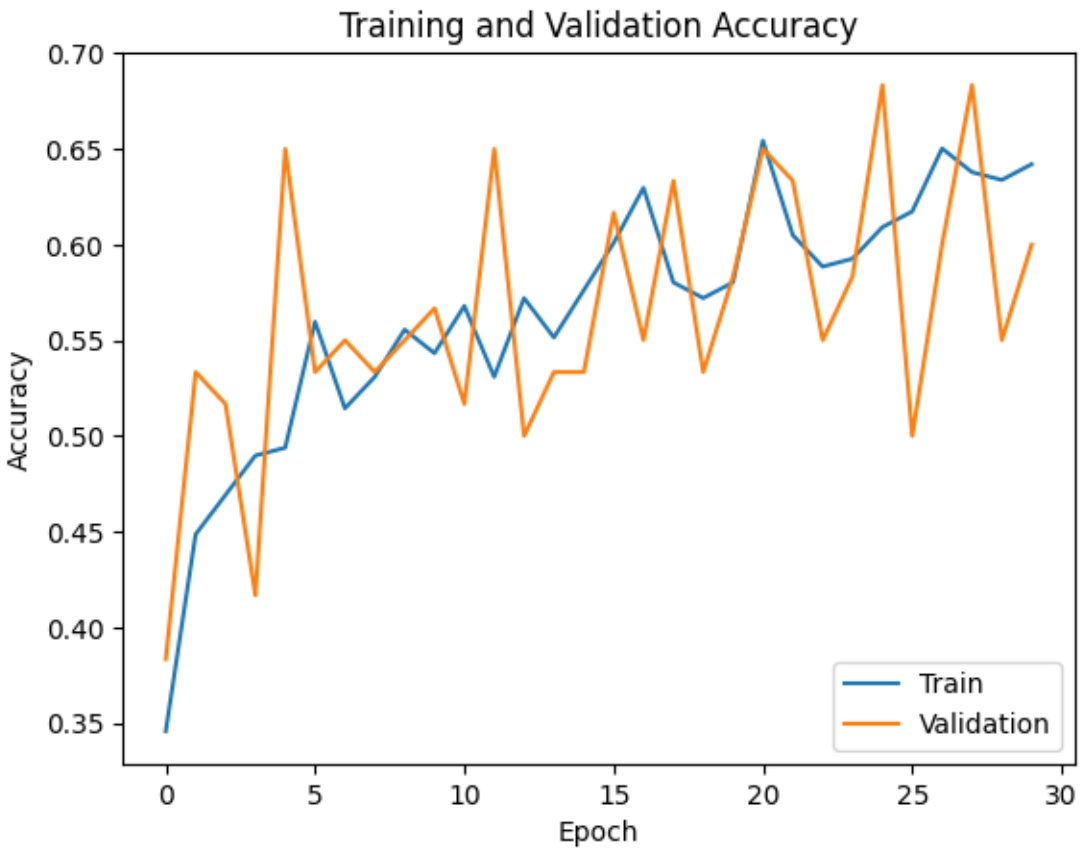
```
8/8 [==============================] - 14s 2s/step - loss: 1.0028 - accuracy:
0.5144 - val_loss: 0.9482 - val_accuracy: 0.5500
Epoch 5/30
8/8 [==============================] - 14s 2s/step - loss: 0.9743 - accuracy:
0.5144 - val_loss: 0.9436 - val_accuracy: 0.6000
Epoch 6/30
8/8 [==============================] - 14s 2s/step - loss: 0.9490 - accuracy:
0.4774 - val_loss: 0.9198 - val_accuracy: 0.5333
Epoch 7/30
8/8 [==============================] - 14s 2s/step - loss: 0.9689 - accuracy:
0.5062 - val_loss: 0.9455 - val_accuracy: 0.4333
Epoch 8/30
8/8 [==============================] - 14s 2s/step - loss: 0.9603 - accuracy:
0.5144 - val_loss: 0.9464 - val_accuracy: 0.4333
Epoch 9/30
8/8 [==============================] - 14s 2s/step - loss: 0.9400 - accuracy:
0.5144 - val_loss: 0.8856 - val_accuracy: 0.6000
Epoch 10/30
8/8 [==============================] - 14s 2s/step - loss: 0.9134 - accuracy:
0.5432 - val_loss: 1.0234 - val_accuracy: 0.4667
Epoch 11/30
8/8 [==============================] - 14s 2s/step - loss: 0.9437 - accuracy:
0.5021 - val_loss: 0.9353 - val_accuracy: 0.6000
Epoch 12/30
8/8 [==============================] - 14s 2s/step - loss: 0.9163 - accuracy:
0.5679 - val_loss: 0.9949 - val_accuracy: 0.5000
Epoch 13/30
8/8 [==============================] - 14s 2s/step - loss: 0.9124 - accuracy:
0.5679 - val_loss: 0.9500 - val_accuracy: 0.4667
Epoch 14/30
8/8 [==============================] - 14s 2s/step - loss: 0.9137 - accuracy:
0.5761 - val_loss: 0.8719 - val_accuracy: 0.5833
Epoch 15/30
8/8 [==============================] - 14s 2s/step - loss: 0.8913 - accuracy:
0.5885 - val_loss: 0.8295 - val_accuracy: 0.6333
Epoch 16/30
8/8 [==============================] - 14s 2s/step - loss: 0.8823 - accuracy:
0.5885 - val_loss: 0.9232 - val_accuracy: 0.6167
Epoch 17/30
8/8 [==============================] - 14s 2s/step - loss: 0.9183 - accuracy:
0.6008 - val_loss: 0.8547 - val_accuracy: 0.6500
Epoch 18/30
8/8 [==============================] - 14s 2s/step - loss: 0.8240 - accuracy:
0.6379 - val_loss: 0.7794 - val_accuracy: 0.6500
Epoch 19/30
8/8 [==============================] - 14s 2s/step - loss: 0.8337 - accuracy:
0.6091 - val_loss: 0.9490 - val_accuracy: 0.5000
Epoch 20/30
```

```
8/8 [==============================] - 14s 2s/step - loss: 0.8737 - accuracy:
0.5679 - val_loss: 0.8986 - val_accuracy: 0.6667
Epoch 21/30
8/8 [==============================] - 15s 2s/step - loss: 0.8322 - accuracy:
0.6132 - val_loss: 0.8089 - val_accuracy: 0.6667
Epoch 22/30
8/8 [==============================] - 15s 2s/step - loss: 0.8722 - accuracy:
0.5638 - val_loss: 0.7533 - val_accuracy: 0.6833
Epoch 23/30
8/8 [==============================] - 15s 2s/step - loss: 0.8195 - accuracy:
0.6379 - val_loss: 0.7725 - val_accuracy: 0.6833
Epoch 24/30
8/8 [==============================] - 15s 2s/step - loss: 0.8265 - accuracy:
0.6420 - val_loss: 0.9660 - val_accuracy: 0.5333
Epoch 25/30
8/8 [==============================] - 15s 2s/step - loss: 0.8695 - accuracy:
0.5802 - val_loss: 0.8012 - val_accuracy: 0.7000
Epoch 26/30
8/8 [==============================] - 15s 2s/step - loss: 0.8168 - accuracy:
0.6461 - val_loss: 0.8328 - val_accuracy: 0.5833
Epoch 27/30
8/8 [==============================] - 17s 2s/step - loss: 0.8098 - accuracy:
0.6667 - val_loss: 0.9367 - val_accuracy: 0.5667
Epoch 28/30
8/8 [==============================] - 14s 2s/step - loss: 0.8389 - accuracy:
0.6461 - val_loss: 0.7552 - val_accuracy: 0.7167
Epoch 29/30
8/8 [==============================] - 14s 2s/step - loss: 0.8054 - accuracy:
0.6626 - val_loss: 0.9065 - val_accuracy: 0.5500
Epoch 30/30
8/8 [==============================] - 14s 2s/step - loss: 0.8459 - accuracy:
0.5720 - val_loss: 0.8214 - val_accuracy: 0.6500
```

[30]:
```
<keras.callbacks.History at 0x7f993981ca00>
```

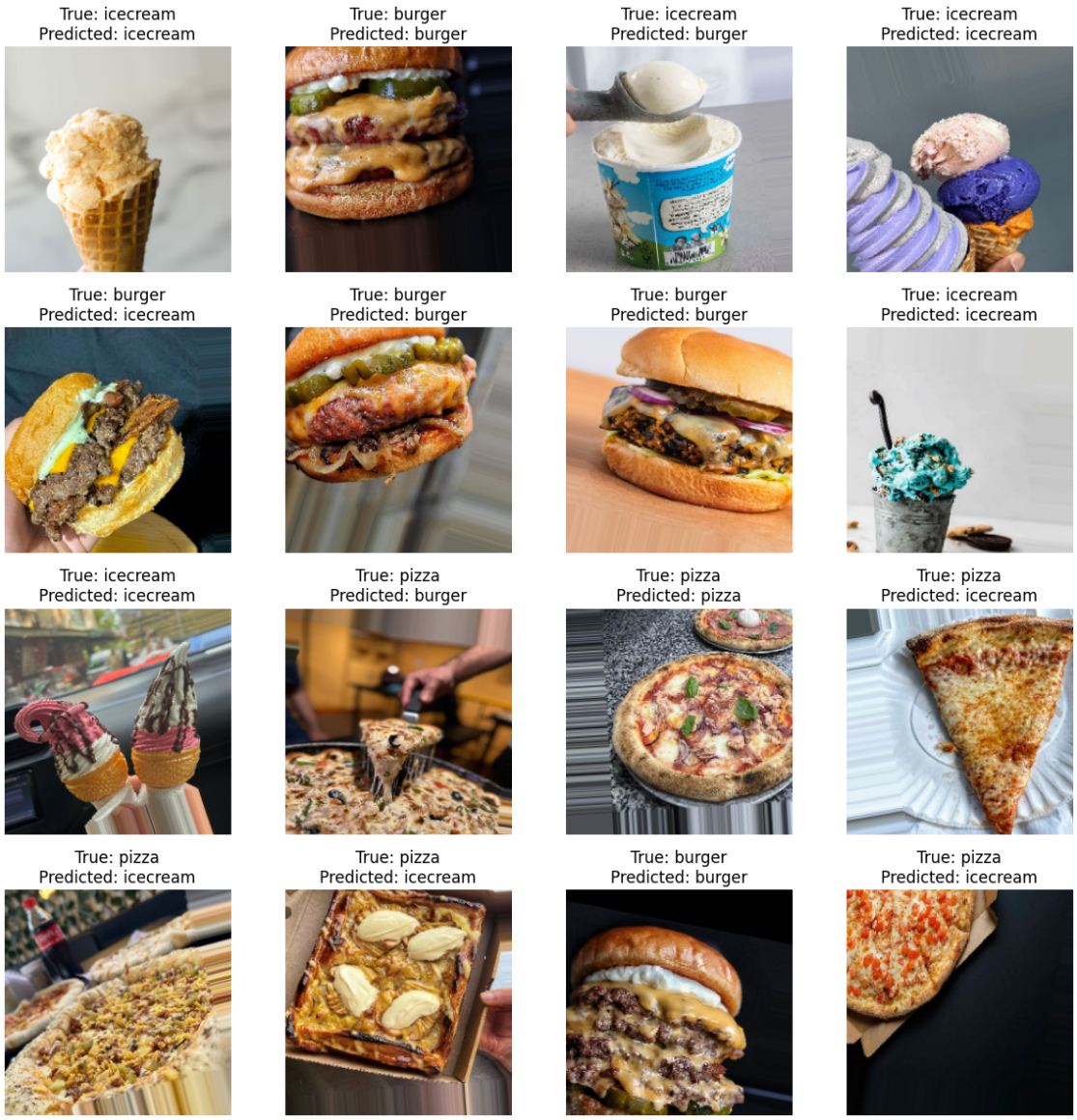
[37]:
```python
import matplotlib.pyplot as plt

# Plot training and validation accuracy
plt.plot(model.history.history['accuracy'])
plt.plot(model.history.history['val_accuracy'])
plt.title('Training and Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(['Train', 'Validation'])
plt.show()

# Plot training and validation loss
```

```
plt.plot(model.history.history['loss'])
plt.plot(model.history.history['val_loss'])
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(['Train', 'Validation'])
plt.show()

# Generate sample predictions
sample_images, sample_labels = next(validation_generator)
sample_predictions = model.predict(sample_images)

# Convert one-hot encoded labels back to class names
class_names = list(train_generator.class_indices.keys())
sample_labels = np.argmax(sample_labels, axis=1)
sample_predictions = np.argmax(sample_predictions, axis=1)

# Display sample images with their true and predicted labels
fig, axes = plt.subplots(4, 4, figsize=(12, 12))
axes = axes.flatten()

for i, ax in enumerate(axes):
    ax.imshow(sample_images[i])
    ax.set_title(f'True: {class_names[sample_labels[i]]}\nPredicted:␣
 ↪{class_names[sample_predictions[i]]}')
    ax.axis('off')

plt.tight_layout()
plt.show()
```


Training and Validation Accuracy
Accuracy
Epoch
Train
Validation
0.70
0.65
0.60
0.55
0.50
0.45
0.40
0.35
0
5
10
15
20
25
30


Training and Validation Loss
Train
Validation
Loss
Epoch
1.2
1.1
1.0
0.9
0.8
0
5
10
15
20
25
30

```
1/1 [==============================] - 9s 9s/step
```




---

# 5 DenseNET

## 5.1 Applying DenseNET for Transfer Learning

[5]:
```
from tensorflow.keras.applications import DenseNet121
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import GlobalAveragePooling2D, Dense
```

[6]:
```
train_generator = datagen.flow_from_directory(
    base_path,
    target_size=image_size,
    batch_size=batch_size,
    class_mode='categorical',
    subset='training',
    shuffle=True
)

validation_generator = datagen.flow_from_directory(
    base_path,
    target_size=image_size,
    batch_size=batch_size,
    class_mode='categorical',
    subset='validation',
    shuffle=True
)
```

```
Found 243 images belonging to 3 classes.
Found 60 images belonging to 3 classes.
```

[7]:
```
base_model = DenseNet121(include_top=False, weights='imagenet',␣
 ↪input_shape=(224, 224, 3))

for layer in base_model.layers:
    layer.trainable = False
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/densenet/densenet121_weights_tf_dim_ordering_tf_kernels_notop.h5
29084464/29084464 [==============================] - 0s 0us/step
```

[8]:
```
model = Sequential()
model.add(base_model)
model.add(GlobalAveragePooling2D())
model.add(Dense(256, activation='relu'))
model.add(Dense(3, activation='softmax'))
```

[9]:
```
model.compile(optimizer='adam', loss='categorical_crossentropy',␣
 ↪metrics=['accuracy'])
model.fit(train_generator, validation_data=validation_generator, epochs=30)
```

```
Epoch 1/30
8/8 [==============================] - 80s 9s/step - loss: 0.8306 - accuracy:
0.6872 - val_loss: 0.3507 - val_accuracy: 0.8833
Epoch 2/30
8/8 [==============================] - 14s 2s/step - loss: 0.1551 - accuracy:
0.9588 - val_loss: 0.2572 - val_accuracy: 0.9000
Epoch 3/30
```

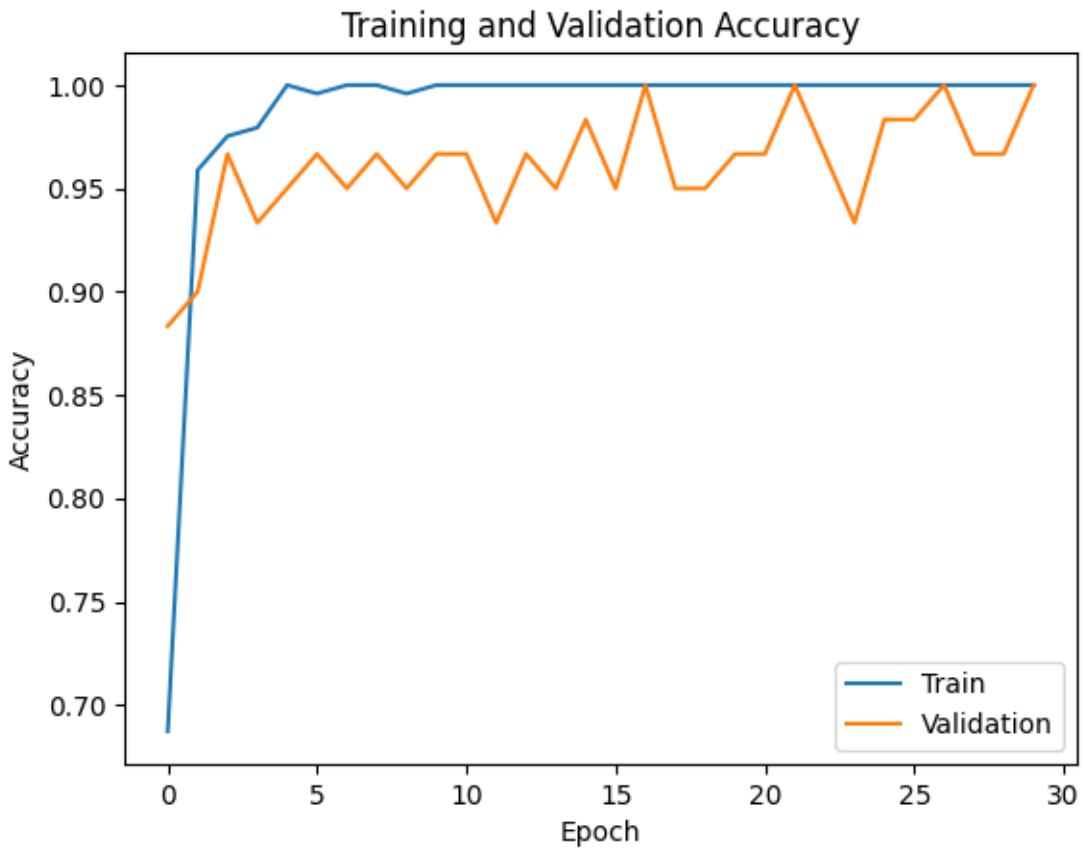
```
8/8 [==============================] - 14s 2s/step - loss: 0.0683 - accuracy:
0.9753 - val_loss: 0.0996 - val_accuracy: 0.9667
Epoch 4/30
8/8 [==============================] - 14s 2s/step - loss: 0.0486 - accuracy:
0.9794 - val_loss: 0.1886 - val_accuracy: 0.9333
Epoch 5/30
8/8 [==============================] - 14s 2s/step - loss: 0.0194 - accuracy:
1.0000 - val_loss: 0.1176 - val_accuracy: 0.9500
Epoch 6/30
8/8 [==============================] - 14s 2s/step - loss: 0.0169 - accuracy:
0.9959 - val_loss: 0.0981 - val_accuracy: 0.9667
Epoch 7/30
8/8 [==============================] - 14s 2s/step - loss: 0.0073 - accuracy:
1.0000 - val_loss: 0.0711 - val_accuracy: 0.9500
Epoch 8/30
8/8 [==============================] - 14s 2s/step - loss: 0.0082 - accuracy:
1.0000 - val_loss: 0.0830 - val_accuracy: 0.9667
Epoch 9/30
8/8 [==============================] - 14s 2s/step - loss: 0.0112 - accuracy:
0.9959 - val_loss: 0.1732 - val_accuracy: 0.9500
Epoch 10/30
8/8 [==============================] - 14s 2s/step - loss: 0.0131 - accuracy:
1.0000 - val_loss: 0.0553 - val_accuracy: 0.9667
Epoch 11/30
8/8 [==============================] - 14s 2s/step - loss: 0.0086 - accuracy:
1.0000 - val_loss: 0.0687 - val_accuracy: 0.9667
Epoch 12/30
8/8 [==============================] - 14s 2s/step - loss: 0.0091 - accuracy:
1.0000 - val_loss: 0.1395 - val_accuracy: 0.9333
Epoch 13/30
8/8 [==============================] - 14s 2s/step - loss: 0.0080 - accuracy:
1.0000 - val_loss: 0.0636 - val_accuracy: 0.9667
Epoch 14/30
8/8 [==============================] - 14s 2s/step - loss: 0.0040 - accuracy:
1.0000 - val_loss: 0.1080 - val_accuracy: 0.9500
Epoch 15/30
8/8 [==============================] - 14s 2s/step - loss: 0.0029 - accuracy:
1.0000 - val_loss: 0.0471 - val_accuracy: 0.9833
Epoch 16/30
8/8 [==============================] - 14s 2s/step - loss: 0.0036 - accuracy:
1.0000 - val_loss: 0.0744 - val_accuracy: 0.9500
Epoch 17/30
8/8 [==============================] - 14s 2s/step - loss: 0.0031 - accuracy:
1.0000 - val_loss: 0.0256 - val_accuracy: 1.0000
Epoch 18/30
8/8 [==============================] - 14s 2s/step - loss: 0.0037 - accuracy:
1.0000 - val_loss: 0.1042 - val_accuracy: 0.9500
Epoch 19/30
```

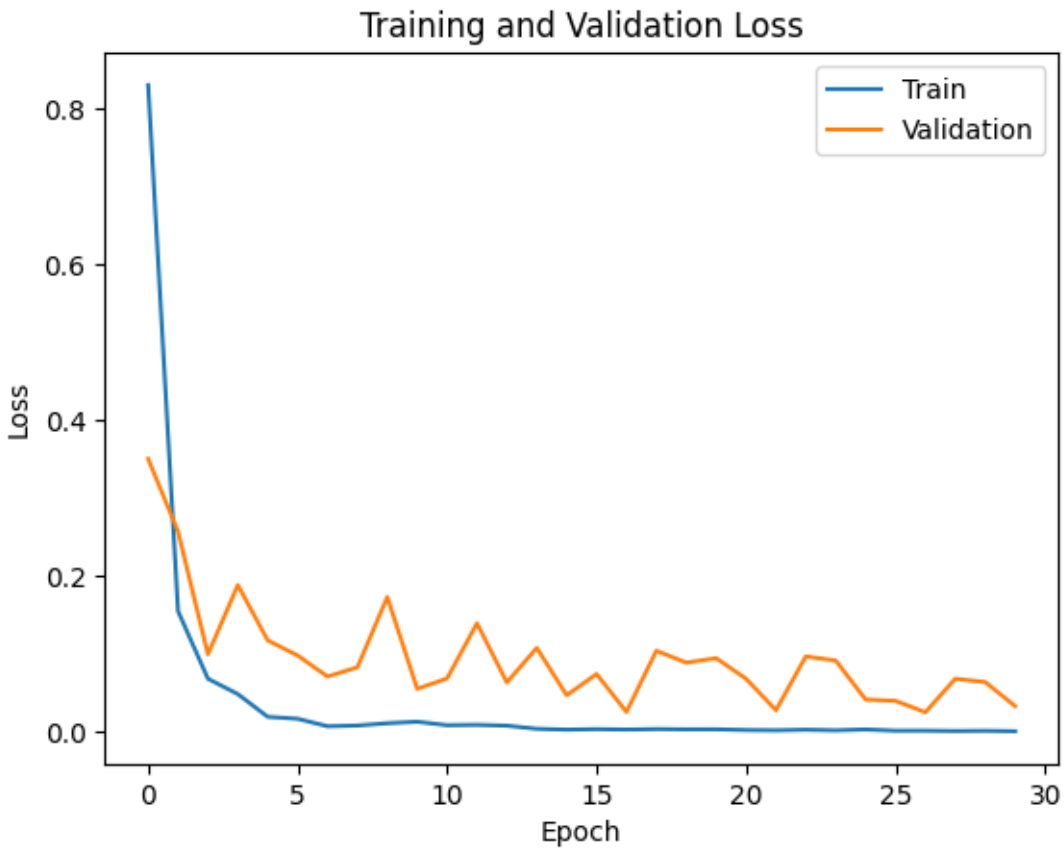
```
8/8 [==============================] - 14s 2s/step - loss: 0.0033 - accuracy:
1.0000 - val_loss: 0.0890 - val_accuracy: 0.9500
Epoch 20/30
8/8 [==============================] - 14s 2s/step - loss: 0.0034 - accuracy:
1.0000 - val_loss: 0.0947 - val_accuracy: 0.9667
Epoch 21/30
8/8 [==============================] - 14s 2s/step - loss: 0.0024 - accuracy:
1.0000 - val_loss: 0.0684 - val_accuracy: 0.9667
Epoch 22/30
8/8 [==============================] - 14s 2s/step - loss: 0.0021 - accuracy:
1.0000 - val_loss: 0.0279 - val_accuracy: 1.0000
Epoch 23/30
8/8 [==============================] - 14s 2s/step - loss: 0.0028 - accuracy:
1.0000 - val_loss: 0.0969 - val_accuracy: 0.9667
Epoch 24/30
8/8 [==============================] - 14s 2s/step - loss: 0.0021 - accuracy:
1.0000 - val_loss: 0.0917 - val_accuracy: 0.9333
Epoch 25/30
8/8 [==============================] - 18s 2s/step - loss: 0.0032 - accuracy:
1.0000 - val_loss: 0.0416 - val_accuracy: 0.9833
Epoch 26/30
8/8 [==============================] - 14s 2s/step - loss: 0.0016 - accuracy:
1.0000 - val_loss: 0.0399 - val_accuracy: 0.9833
Epoch 27/30
8/8 [==============================] - 14s 2s/step - loss: 0.0016 - accuracy:
1.0000 - val_loss: 0.0253 - val_accuracy: 1.0000
Epoch 28/30
8/8 [==============================] - 14s 2s/step - loss: 0.0012 - accuracy:
1.0000 - val_loss: 0.0683 - val_accuracy: 0.9667
Epoch 29/30
8/8 [==============================] - 15s 2s/step - loss: 0.0015 - accuracy:
1.0000 - val_loss: 0.0641 - val_accuracy: 0.9667
Epoch 30/30
8/8 [==============================] - 14s 2s/step - loss: 9.1177e-04 -
accuracy: 1.0000 - val_loss: 0.0332 - val_accuracy: 1.0000
```

[9]:
```
<keras.callbacks.History at 0x7f99c0190670>
```

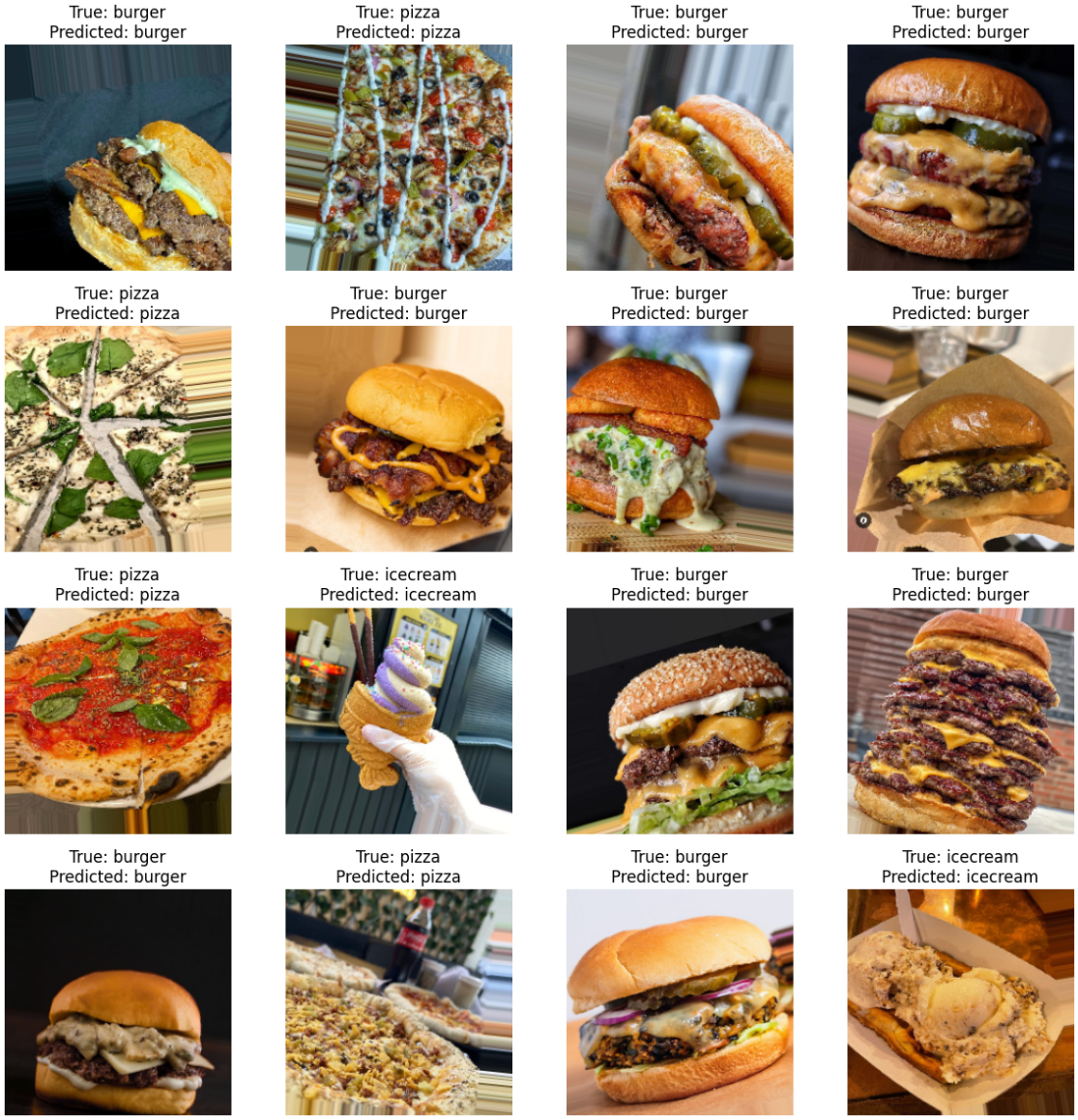
[10]:
```python
import matplotlib.pyplot as plt

# Plot training and validation accuracy
plt.plot(model.history.history['accuracy'])
plt.plot(model.history.history['val_accuracy'])
plt.title('Training and Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(['Train', 'Validation'])
```

```
plt.show()

# Plot training and validation loss
plt.plot(model.history.history['loss'])
plt.plot(model.history.history['val_loss'])
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(['Train', 'Validation'])
plt.show()

# Generate sample predictions
sample_images, sample_labels = next(validation_generator)
sample_predictions = model.predict(sample_images)

# Convert one-hot encoded labels back to class names
class_names = list(train_generator.class_indices.keys())
sample_labels = np.argmax(sample_labels, axis=1)
sample_predictions = np.argmax(sample_predictions, axis=1)

# Display sample images with their true and predicted labels
fig, axes = plt.subplots(4, 4, figsize=(12, 12))
axes = axes.flatten()

for i, ax in enumerate(axes):
    ax.imshow(sample_images[i])
    ax.set_title(f'True: {class_names[sample_labels[i]]}\nPredicted:␣
 ↪{class_names[sample_predictions[i]]}')
    ax.axis('off')

plt.tight_layout()
plt.show()
```


Training and Validation Accuracy
1.00
0.95
0.90
0.85
0.80
0.75
0.70
Accuracy
0
5
10
15
20
25
30
Epoch
Train
Validation


Training and Validation Loss
Train
Validation
Loss
Epoch
0.0
0.2
0.4
0.6
0.8
0
5
10
15
20
25
30

```
1/1 [==============================] - 2s 2s/step
```




---

# 6 MobileNetV2

## 6.1 Applying MobileNetV2 for Transfer Learning

[11]:
```
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import GlobalAveragePooling2D, Dense
```

[12]:
```
train_generator = datagen.flow_from_directory(
    base_path,
    target_size=image_size,
    batch_size=batch_size,
    class_mode='categorical',
    subset='training',
    shuffle=True
)

validation_generator = datagen.flow_from_directory(
    base_path,
    target_size=image_size,
    batch_size=batch_size,
    class_mode='categorical',
    subset='validation',
    shuffle=True
)
```

```
Found 243 images belonging to 3 classes.
Found 60 images belonging to 3 classes.
```

[13]:
```
base_model = MobileNetV2(include_top=False, weights='imagenet',␣
 ↪input_shape=(224, 224, 3))

for layer in base_model.layers:
    layer.trainable = False
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applicatio
ns/mobilenet_v2/mobilenet_v2_weights_tf_dim_ordering_tf_kernels_1.0_224_no_top.h
5
9406464/9406464 [==============================] - 0s 0us/step
```

[14]:
```
model = Sequential()
model.add(base_model)
model.add(GlobalAveragePooling2D())
model.add(Dense(256, activation='relu'))
model.add(Dense(3, activation='softmax'))
```

[16]:
```
model.compile(optimizer='adam', loss='categorical_crossentropy',␣
 ↪metrics=['accuracy'])
model.fit(train_generator, validation_data=validation_generator, epochs=30)
```

```
Epoch 1/30
8/8 [==============================] - 20s 2s/step - loss: 0.6556 - accuracy:
0.7737 - val_loss: 0.1670 - val_accuracy: 0.9000
Epoch 2/30
8/8 [==============================] - 14s 2s/step - loss: 0.0482 - accuracy:
0.9877 - val_loss: 0.0697 - val_accuracy: 0.9833
```

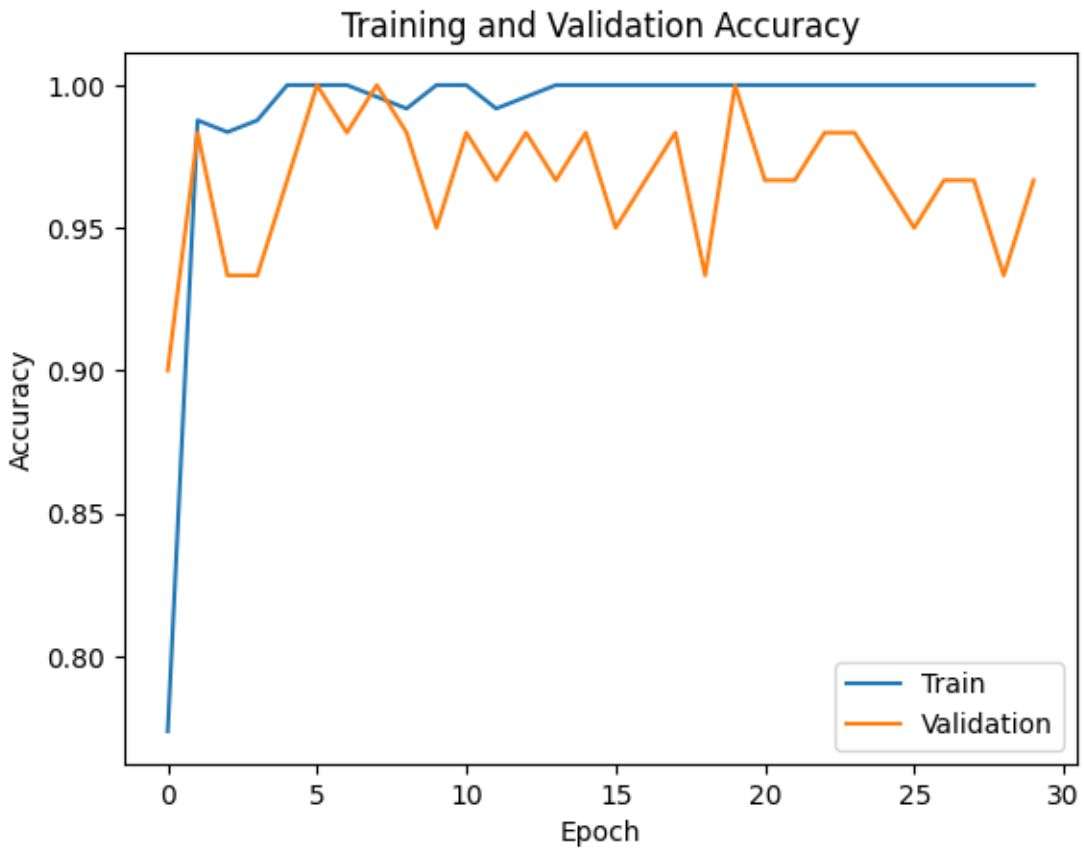
```
Epoch 3/30
8/8 [==============================] - 14s 2s/step - loss: 0.0380 - accuracy:
0.9835 - val_loss: 0.2025 - val_accuracy: 0.9333
Epoch 4/30
8/8 [==============================] - 14s 2s/step - loss: 0.0285 - accuracy:
0.9877 - val_loss: 0.1394 - val_accuracy: 0.9333
Epoch 5/30
8/8 [==============================] - 14s 2s/step - loss: 0.0071 - accuracy:
1.0000 - val_loss: 0.0526 - val_accuracy: 0.9667
Epoch 6/30
8/8 [==============================] - 14s 2s/step - loss: 0.0058 - accuracy:
1.0000 - val_loss: 0.0308 - val_accuracy: 1.0000
Epoch 7/30
8/8 [==============================] - 15s 2s/step - loss: 0.0040 - accuracy:
1.0000 - val_loss: 0.0642 - val_accuracy: 0.9833
Epoch 8/30
8/8 [==============================] - 14s 2s/step - loss: 0.0119 - accuracy:
0.9959 - val_loss: 0.0289 - val_accuracy: 1.0000
Epoch 9/30
8/8 [==============================] - 14s 2s/step - loss: 0.0116 - accuracy:
0.9918 - val_loss: 0.0312 - val_accuracy: 0.9833
Epoch 10/30
8/8 [==============================] - 14s 2s/step - loss: 0.0070 - accuracy:
1.0000 - val_loss: 0.1224 - val_accuracy: 0.9500
Epoch 11/30
8/8 [==============================] - 14s 2s/step - loss: 0.0031 - accuracy:
1.0000 - val_loss: 0.0474 - val_accuracy: 0.9833
Epoch 12/30
8/8 [==============================] - 14s 2s/step - loss: 0.0106 - accuracy:
0.9918 - val_loss: 0.1647 - val_accuracy: 0.9667
Epoch 13/30
8/8 [==============================] - 16s 2s/step - loss: 0.0047 - accuracy:
0.9959 - val_loss: 0.0612 - val_accuracy: 0.9833
Epoch 14/30
8/8 [==============================] - 14s 2s/step - loss: 0.0034 - accuracy:
1.0000 - val_loss: 0.0958 - val_accuracy: 0.9667
Epoch 15/30
8/8 [==============================] - 14s 2s/step - loss: 0.0032 - accuracy:
1.0000 - val_loss: 0.0872 - val_accuracy: 0.9833
Epoch 16/30
8/8 [==============================] - 14s 2s/step - loss: 0.0013 - accuracy:
1.0000 - val_loss: 0.1019 - val_accuracy: 0.9500
Epoch 17/30
8/8 [==============================] - 14s 2s/step - loss: 0.0011 - accuracy:
1.0000 - val_loss: 0.1264 - val_accuracy: 0.9667
Epoch 18/30
8/8 [==============================] - 14s 2s/step - loss: 0.0013 - accuracy:
1.0000 - val_loss: 0.0710 - val_accuracy: 0.9833
```

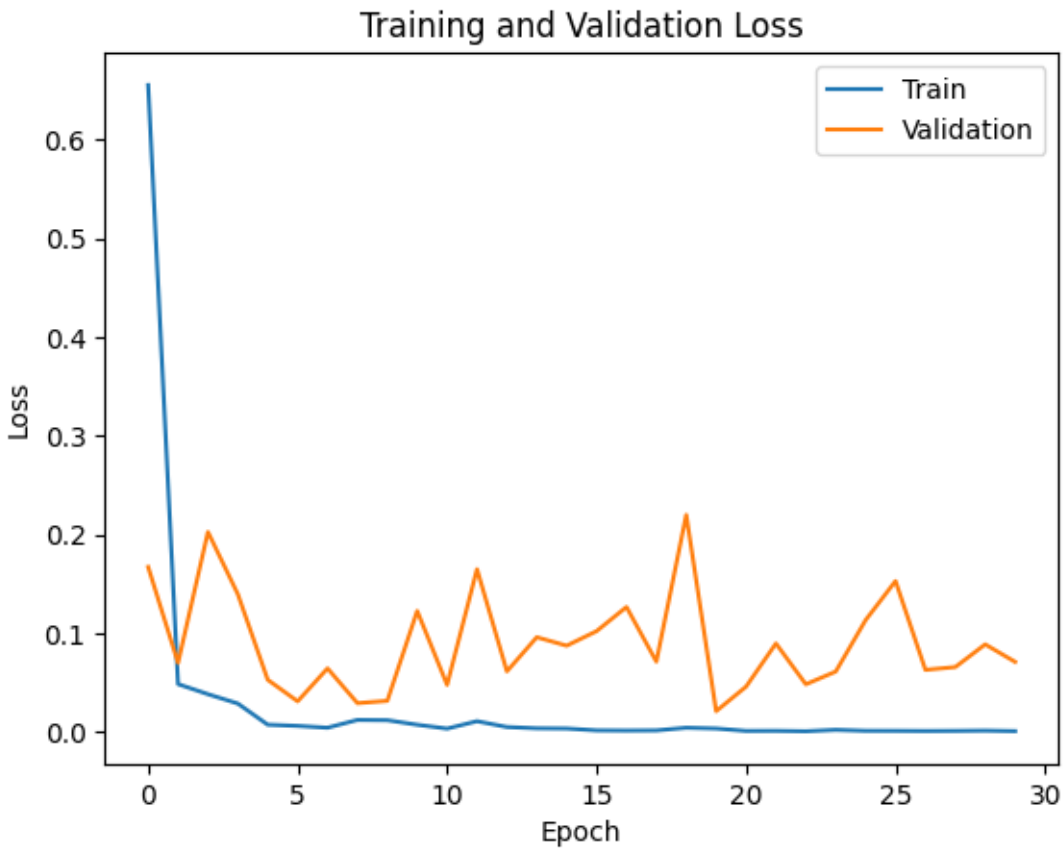
```
Epoch 19/30
8/8 [==============================] - 14s 2s/step - loss: 0.0040 - accuracy:
1.0000 - val_loss: 0.2199 - val_accuracy: 0.9333
Epoch 20/30
8/8 [==============================] - 14s 2s/step - loss: 0.0032 - accuracy:
1.0000 - val_loss: 0.0208 - val_accuracy: 1.0000
Epoch 21/30
8/8 [==============================] - 14s 2s/step - loss: 7.7839e-04 -
accuracy: 1.0000 - val_loss: 0.0456 - val_accuracy: 0.9667
Epoch 22/30
8/8 [==============================] - 14s 2s/step - loss: 8.4559e-04 -
accuracy: 1.0000 - val_loss: 0.0894 - val_accuracy: 0.9667
Epoch 23/30
8/8 [==============================] - 14s 2s/step - loss: 4.4549e-04 -
accuracy: 1.0000 - val_loss: 0.0480 - val_accuracy: 0.9833
Epoch 24/30
8/8 [==============================] - 14s 2s/step - loss: 0.0018 - accuracy:
1.0000 - val_loss: 0.0609 - val_accuracy: 0.9833
Epoch 25/30
8/8 [==============================] - 14s 2s/step - loss: 8.9169e-04 -
accuracy: 1.0000 - val_loss: 0.1131 - val_accuracy: 0.9667
Epoch 26/30
8/8 [==============================] - 14s 2s/step - loss: 8.4760e-04 -
accuracy: 1.0000 - val_loss: 0.1527 - val_accuracy: 0.9500
Epoch 27/30
8/8 [==============================] - 14s 2s/step - loss: 6.9664e-04 -
accuracy: 1.0000 - val_loss: 0.0627 - val_accuracy: 0.9667
Epoch 28/30
8/8 [==============================] - 14s 2s/step - loss: 8.1388e-04 -
accuracy: 1.0000 - val_loss: 0.0655 - val_accuracy: 0.9667
Epoch 29/30
8/8 [==============================] - 14s 2s/step - loss: 0.0011 - accuracy:
1.0000 - val_loss: 0.0885 - val_accuracy: 0.9333
Epoch 30/30
8/8 [==============================] - 14s 2s/step - loss: 5.7323e-04 -
accuracy: 1.0000 - val_loss: 0.0708 - val_accuracy: 0.9667
```

[16]:
```
<keras.callbacks.History at 0x7f99341e6c80>
```

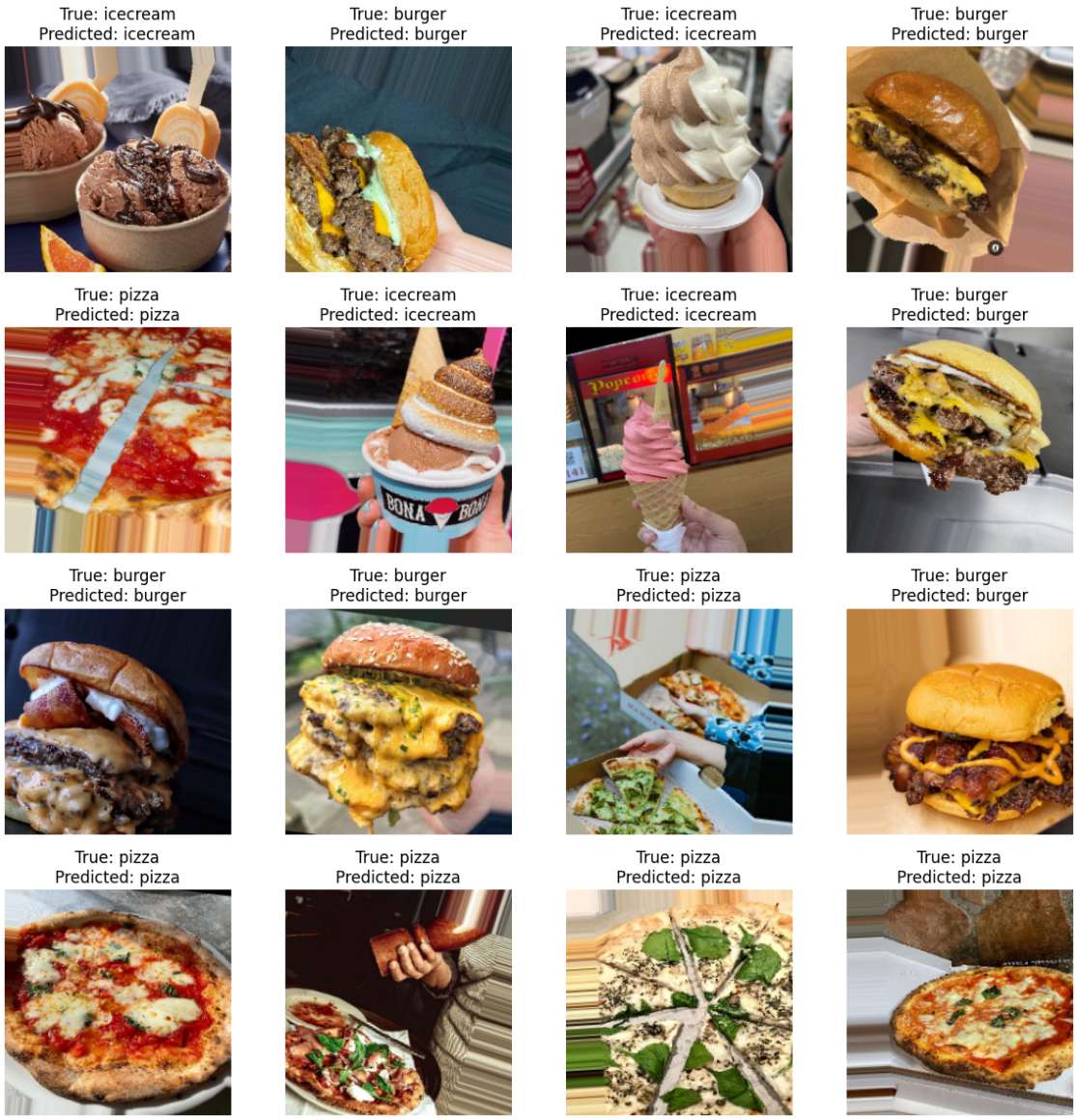
[17]:
```python
import matplotlib.pyplot as plt

# Plot training and validation accuracy
plt.plot(model.history.history['accuracy'])
plt.plot(model.history.history['val_accuracy'])
plt.title('Training and Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
```

```
plt.legend(['Train', 'Validation'])
plt.show()

# Plot training and validation loss
plt.plot(model.history.history['loss'])
plt.plot(model.history.history['val_loss'])
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(['Train', 'Validation'])
plt.show()

# Generate sample predictions
sample_images, sample_labels = next(validation_generator)
sample_predictions = model.predict(sample_images)

# Convert one-hot encoded labels back to class names
class_names = list(train_generator.class_indices.keys())
sample_labels = np.argmax(sample_labels, axis=1)
sample_predictions = np.argmax(sample_predictions, axis=1)

# Display sample images with their true and predicted labels
fig, axes = plt.subplots(4, 4, figsize=(12, 12))
axes = axes.flatten()

for i, ax in enumerate(axes):
    ax.imshow(sample_images[i])
    ax.set_title(f'True: {class_names[sample_labels[i]]}\nPredicted:␣
 ↪{class_names[sample_predictions[i]]}')
    ax.axis('off')

plt.tight_layout()
plt.show()
```


Training and Validation Accuracy
Accuracy
Epoch
Train
Validation
1.00
0.95
0.90
0.85
0.80
0
5
10
15
20
25
30




```
1/1 [==============================] - 1s 760ms/step
```




---

# 7 EfficientNet

## 7.1 Applying EfficientNet for Transfer Learning

[18]:
```
!pip install efficientnet
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-
wheels/public/simple/
Collecting efficientnet
  Downloading efficientnet-1.1.1-py3-none-any.whl (18 kB)
```

```
Collecting keras-applications<=1.0.8,>=1.0.7 (from efficientnet)
  Downloading Keras_Applications-1.0.8-py3-none-any.whl (50 kB)
                              50.7/50.7 kB
3.4 MB/s eta 0:00:00
Requirement already satisfied: scikit-image in
/usr/local/lib/python3.10/dist-packages (from efficientnet) (0.19.3)
Requirement already satisfied: numpy>=1.9.1 in /usr/local/lib/python3.10/dist-
packages (from keras-applications<=1.0.8,>=1.0.7->efficientnet) (1.22.4)
Requirement already satisfied: h5py in /usr/local/lib/python3.10/dist-packages
(from keras-applications<=1.0.8,>=1.0.7->efficientnet) (3.8.0)
Requirement already satisfied: scipy>=1.4.1 in /usr/local/lib/python3.10/dist-
packages (from scikit-image->efficientnet) (1.10.1)
Requirement already satisfied: networkx>=2.2 in /usr/local/lib/python3.10/dist-
packages (from scikit-image->efficientnet) (3.1)
Requirement already satisfied: pillow!=7.1.0,!=7.1.1,!=8.3.0,>=6.1.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-image->efficientnet)
(8.4.0)
Requirement already satisfied: imageio>=2.4.1 in /usr/local/lib/python3.10/dist-
packages (from scikit-image->efficientnet) (2.25.1)
Requirement already satisfied: tifffile>=2019.7.26 in
/usr/local/lib/python3.10/dist-packages (from scikit-image->efficientnet)
(2023.4.12)
Requirement already satisfied: PyWavelets>=1.1.1 in
/usr/local/lib/python3.10/dist-packages (from scikit-image->efficientnet)
(1.4.1)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-image->efficientnet) (23.1)
Installing collected packages: keras-applications, efficientnet
Successfully installed efficientnet-1.1.1 keras-applications-1.0.8
```

[19]:
```python
from efficientnet.tfkeras import EfficientNetB0
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import GlobalAveragePooling2D, Dense
```

[20]:
```python
train_generator = datagen.flow_from_directory(
    base_path,
    target_size=image_size,
    batch_size=batch_size,
    class_mode='categorical',
    subset='training',
    shuffle=True
)

validation_generator = datagen.flow_from_directory(
    base_path,
    target_size=image_size,
    batch_size=batch_size,
```

```
    class_mode='categorical',
    subset='validation',
    shuffle=True
)
```

```
Found 243 images belonging to 3 classes.
Found 60 images belonging to 3 classes.
```

[21]:
```
base_model = EfficientNetB0(include_top=False, weights='imagenet',␣
 ↪input_shape=(224, 224, 3))

for layer in base_model.layers:
    layer.trainable = False
```

```
Downloading data from https://github.com/Callidior/keras-applications/releases/d
ownload/efficientnet/efficientnet-b0_weights_tf_dim_ordering_tf_kernels_autoaugm
ent_notop.h5
16804768/16804768 [==============================] - 0s 0us/step
```

[22]:
```
model = Sequential()
model.add(base_model)
model.add(GlobalAveragePooling2D())
model.add(Dense(256, activation='relu'))
model.add(Dense(3, activation='softmax'))
```

[23]:
```
model.compile(optimizer='adam', loss='categorical_crossentropy',␣
 ↪metrics=['accuracy'])
model.fit(train_generator, validation_data=validation_generator, epochs=30)
```

```
Epoch 1/30
8/8 [==============================] - 27s 2s/step - loss: 0.4566 - accuracy:
0.8354 - val_loss: 0.1288 - val_accuracy: 0.9500
Epoch 2/30
8/8 [==============================] - 14s 2s/step - loss: 0.0836 - accuracy:
0.9547 - val_loss: 0.0633 - val_accuracy: 0.9833
Epoch 3/30
8/8 [==============================] - 16s 2s/step - loss: 0.0288 - accuracy:
0.9918 - val_loss: 0.0703 - val_accuracy: 0.9667
Epoch 4/30
8/8 [==============================] - 14s 2s/step - loss: 0.0170 - accuracy:
0.9959 - val_loss: 0.1053 - val_accuracy: 0.9833
Epoch 5/30
8/8 [==============================] - 14s 2s/step - loss: 0.0057 - accuracy:
1.0000 - val_loss: 0.0760 - val_accuracy: 0.9667
Epoch 6/30
8/8 [==============================] - 14s 2s/step - loss: 0.0178 - accuracy:
0.9959 - val_loss: 0.0580 - val_accuracy: 0.9833
Epoch 7/30
```

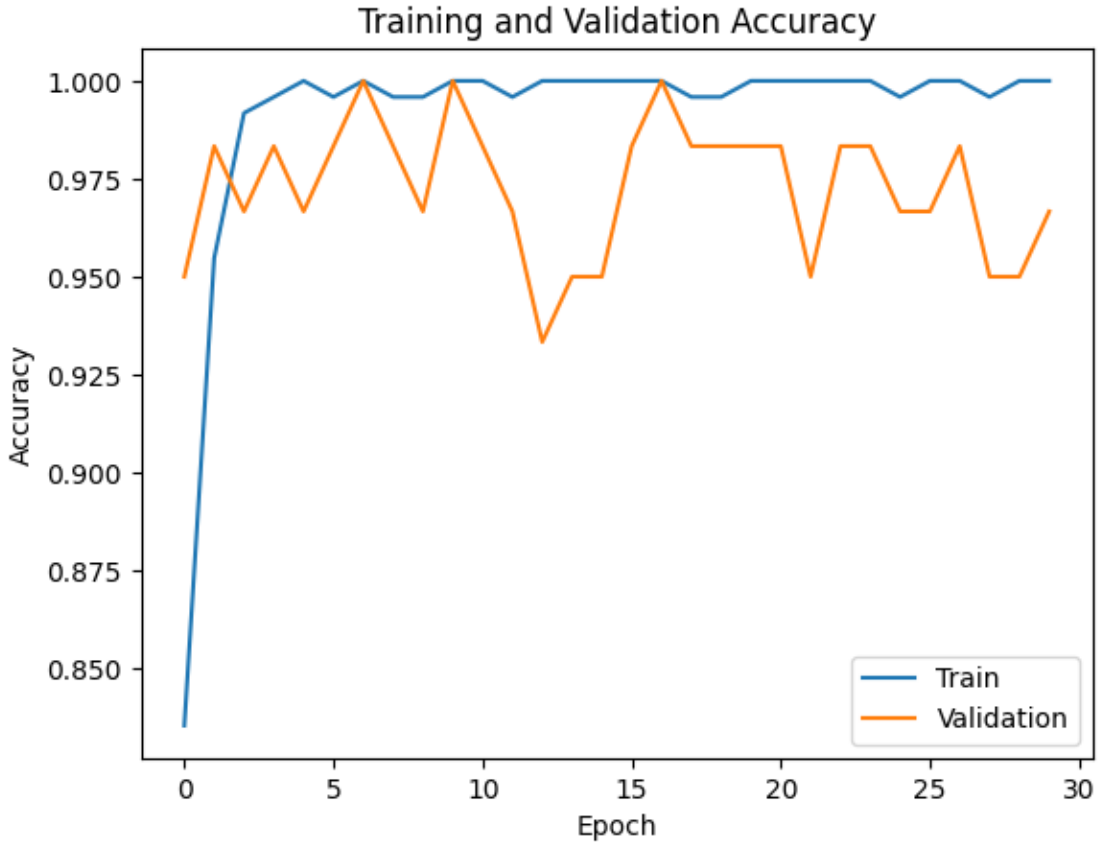
```
8/8 [==============================] - 14s 2s/step - loss: 0.0060 - accuracy:
1.0000 - val_loss: 0.0364 - val_accuracy: 1.0000
Epoch 8/30
8/8 [==============================] - 14s 2s/step - loss: 0.0140 - accuracy:
0.9959 - val_loss: 0.0340 - val_accuracy: 0.9833
Epoch 9/30
8/8 [==============================] - 14s 2s/step - loss: 0.0087 - accuracy:
0.9959 - val_loss: 0.0853 - val_accuracy: 0.9667
Epoch 10/30
8/8 [==============================] - 15s 2s/step - loss: 0.0036 - accuracy:
1.0000 - val_loss: 0.0352 - val_accuracy: 1.0000
Epoch 11/30
8/8 [==============================] - 17s 2s/step - loss: 0.0044 - accuracy:
1.0000 - val_loss: 0.0324 - val_accuracy: 0.9833
Epoch 12/30
8/8 [==============================] - 14s 2s/step - loss: 0.0082 - accuracy:
0.9959 - val_loss: 0.0677 - val_accuracy: 0.9667
Epoch 13/30
8/8 [==============================] - 14s 2s/step - loss: 0.0021 - accuracy:
1.0000 - val_loss: 0.1468 - val_accuracy: 0.9333
Epoch 14/30
8/8 [==============================] - 14s 2s/step - loss: 0.0033 - accuracy:
1.0000 - val_loss: 0.1177 - val_accuracy: 0.9500
Epoch 15/30
8/8 [==============================] - 14s 2s/step - loss: 0.0037 - accuracy:
1.0000 - val_loss: 0.1065 - val_accuracy: 0.9500
Epoch 16/30
8/8 [==============================] - 14s 2s/step - loss: 9.7019e-04 -
accuracy: 1.0000 - val_loss: 0.0593 - val_accuracy: 0.9833
Epoch 17/30
8/8 [==============================] - 14s 2s/step - loss: 0.0013 - accuracy:
1.0000 - val_loss: 0.0288 - val_accuracy: 1.0000
Epoch 18/30
8/8 [==============================] - 14s 2s/step - loss: 0.0094 - accuracy:
0.9959 - val_loss: 0.0540 - val_accuracy: 0.9833
Epoch 19/30
8/8 [==============================] - 14s 2s/step - loss: 0.0069 - accuracy:
0.9959 - val_loss: 0.0780 - val_accuracy: 0.9833
Epoch 20/30
8/8 [==============================] - 14s 2s/step - loss: 0.0045 - accuracy:
1.0000 - val_loss: 0.0566 - val_accuracy: 0.9833
Epoch 21/30
8/8 [==============================] - 14s 2s/step - loss: 0.0022 - accuracy:
1.0000 - val_loss: 0.0601 - val_accuracy: 0.9833
Epoch 22/30
8/8 [==============================] - 14s 2s/step - loss: 0.0032 - accuracy:
1.0000 - val_loss: 0.0768 - val_accuracy: 0.9500
Epoch 23/30
```

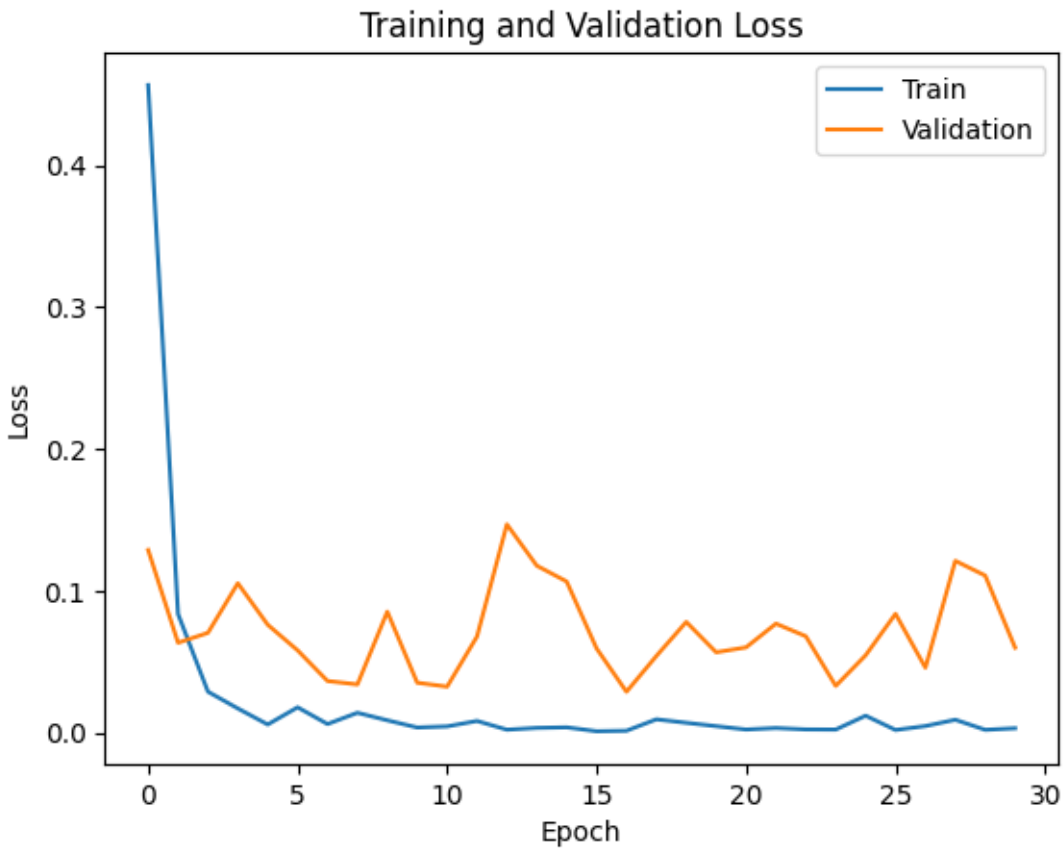
```
8/8 [==============================] - 14s 2s/step - loss: 0.0022 - accuracy:
1.0000 - val_loss: 0.0679 - val_accuracy: 0.9833
Epoch 24/30
8/8 [==============================] - 16s 2s/step - loss: 0.0021 - accuracy:
1.0000 - val_loss: 0.0329 - val_accuracy: 0.9833
Epoch 25/30
8/8 [==============================] - 14s 2s/step - loss: 0.0119 - accuracy:
0.9959 - val_loss: 0.0545 - val_accuracy: 0.9667
Epoch 26/30
8/8 [==============================] - 14s 2s/step - loss: 0.0019 - accuracy:
1.0000 - val_loss: 0.0837 - val_accuracy: 0.9667
Epoch 27/30
8/8 [==============================] - 16s 2s/step - loss: 0.0046 - accuracy:
1.0000 - val_loss: 0.0458 - val_accuracy: 0.9833
Epoch 28/30
8/8 [==============================] - 14s 2s/step - loss: 0.0091 - accuracy:
0.9959 - val_loss: 0.1211 - val_accuracy: 0.9500
Epoch 29/30
8/8 [==============================] - 14s 2s/step - loss: 0.0020 - accuracy:
1.0000 - val_loss: 0.1107 - val_accuracy: 0.9500
Epoch 30/30
8/8 [==============================] - 14s 2s/step - loss: 0.0031 - accuracy:
1.0000 - val_loss: 0.0599 - val_accuracy: 0.9667
```

[23]:
```
<keras.callbacks.History at 0x7f98f42efeb0>
```

[24]:
```python
import matplotlib.pyplot as plt

# Plot training and validation accuracy
plt.plot(model.history.history['accuracy'])
plt.plot(model.history.history['val_accuracy'])
plt.title('Training and Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(['Train', 'Validation'])
plt.show()

# Plot training and validation loss
plt.plot(model.history.history['loss'])
plt.plot(model.history.history['val_loss'])
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(['Train', 'Validation'])
plt.show()

# Generate sample predictions
```

```
sample_images, sample_labels = next(validation_generator)
sample_predictions = model.predict(sample_images)

# Convert one-hot encoded labels back to class names
class_names = list(train_generator.class_indices.keys())
sample_labels = np.argmax(sample_labels, axis=1)
sample_predictions = np.argmax(sample_predictions, axis=1)

# Display sample images with their true and predicted labels
fig, axes = plt.subplots(4, 4, figsize=(12, 12))
axes = axes.flatten()

for i, ax in enumerate(axes):
    ax.imshow(sample_images[i])
    ax.set_title(f'True: {class_names[sample_labels[i]]}\nPredicted:␣
 ↪{class_names[sample_predictions[i]]}')
    ax.axis('off')

plt.tight_layout()
plt.show()
```




Training and Validation Loss
Train
Validation
Loss
0.0
0.1
0.2
0.3
0.4
0
5
10
15
20
25
30
Epoch

```
1/1 [==============================] - 2s 2s/step
```

True: icecream
Predicted: icecream

True: pizza
Predicted: pizza

True: pizza
Predicted: pizza

True: icecream
Predicted: icecream

True: pizza
Predicted: pizza

True: burger
Predicted: burger

True: icecream
Predicted: icecream

True: pizza
Predicted: pizza

True: burger
Predicted: burger

True: pizza
Predicted: icecream

True: icecream
Predicted: icecream

True: burger
Predicted: burger

True: burger
Predicted: burger

True: icecream
Predicted: icecream

True: burger
Predicted: burger

True: icecream
Predicted: icecream