Food 101 kagle data set

Become one with data

import zipfile

!wget https://storage.googleapis.com/ztm\_tf\_course/food\_vision/pizza\_steak.zip

zip\_ref = zipfile.ZipFile("pizza\_steak.zip")

zip\_ref.extractall()

zip\_ref.close()

import os

for dirpaths,dirnames, filenames in os.walk("pizza\_steak"):

print(f"There are {len(dirnames)} directories and {len(filenames)} files in {dirpaths} ")

num\_steak\_images\_train = len(os.listdir("pizza\_steak/train/steak"))

num\_steak\_images\_train

Find classnames

import pathlib

import numpy as np

data\_dir = pathlib.Path("pizza\_steak/train")

class\_names = np.array(sorted([item.name for item in data\_dir.glob("\*")]))

print(class\_names)

Method for Random image from a class

import matplotlib.pyplot as plt

import matplotlib.image as mpimage

import random

def view\_random\_image(target\_dir, target\_class):

target\_folder = target\_dir + target\_class

random\_image= random.sample(os.listdir(target\_folder),1)

#print(random\_image)

image = mpimage.imread(target\_folder+"/"+random\_image[0])

#print(f"Image Details: {image}")

plt.imshow(image)

plt.title(target\_class)

plt.axis("off")

print(f"Image Shape:{image.shape}")

return image

Normalizing Data(Making data between 0 & 1)

img/255.

Graphical user interface, text, application

Description automatically generated

First CNN Model

from tensorflow.keras.preprocessing.image import ImageDataGenerator

import tensorflow as tf

tf.random.set\_seed(42)

train\_Datagen = ImageDataGenerator(rescale=1./255)

valid\_Datagen = ImageDataGenerator(rescale=1./255)

train\_dir = "pizza\_steak/train/"

test\_dir = "pizza\_steak/test/"

train\_data = train\_Datagen.flow\_from\_directory(train\_dir,batch\_size=32,target\_size=(224,224),class\_mode="binary",seed=42)

valid\_data = valid\_Datagen.flow\_from\_directory(test\_dir,batch\_size=32,target\_size=(224,224),class\_mode="binary",seed=42)

model\_1 = tf.keras.models.Sequential([

tf.keras.layers.Conv2D(filters=10,kernel\_size=3,activation="relu",input\_shape=(224,224,3)),

tf.keras.layers.Conv2D(10,3,activation="relu"),

tf.keras.layers.MaxPool2D(pool\_size=2,padding="valid"),

tf.keras.layers.Conv2D(10,3,activation="relu"),

tf.keras.layers.Conv2D(10,3,activation="relu"),

tf.keras.layers.MaxPool2D(2),

tf.keras.layers.Flatten(),

tf.keras.layers.Dense(1,activation="sigmoid")

])

model\_1.compile(loss="binary\_crossentropy",

optimizer=tf.keras.optimizers.Adam(),

metrics=["accuracy"])

history\_1 = model\_1.fit(train\_data,epochs=5,steps\_per\_epoch=len(train\_data),

validation\_data=valid\_data,validation\_steps=len(valid\_data))

CNN Explainer: <https://poloclub.github.io/cnn-explainer/>

model\_1.summary()

Testing old model from classification problem with image Data

import tensorflow as tf

tf.random.set\_seed(42)

model\_2 = tf.keras.models.Sequential([

tf.keras.layers.Flatten(input\_shape=(224,224,3)),

tf.keras.layers.Dense(4,activation="relu"),

tf.keras.layers.Dense(4,activation="relu"),

tf.keras.layers.Dense(1,activation="sigmoid")

])

model\_2.compile(loss="binary\_crossentropy",

optimizer=tf.keras.optimizers.Adam(),

metrics=["accuracy"])

history\_2 = model\_2.fit(train\_data,epochs=5,steps\_per\_epoch=len(train\_data),

validation\_data=valid\_data,validation\_steps=len(valid\_data))

model\_2.summary()

Trying to improve old model with 100 neurons

import tensorflow as tf

tf.random.set\_seed(42)

model\_3 = tf.keras.models.Sequential([

tf.keras.layers.Flatten(input\_shape=(224,224,3)),

tf.keras.layers.Dense(100,activation="relu"),

tf.keras.layers.Dense(100,activation="relu"),

tf.keras.layers.Dense(100,activation="relu"),

tf.keras.layers.Dense(1,activation="sigmoid")

])

model\_3.compile(loss="binary\_crossentropy",

optimizer=tf.keras.optimizers.Adam(),

metrics=["accuracy"])

history\_3 = model\_3.fit(train\_data,epochs=5,steps\_per\_epoch=len(train\_data),

validation\_data=valid\_data,validation\_steps=len(valid\_data))

model\_3.summary()

contains 15millions trainable parameters which is 500 times than model\_1 which is using 31,000 parameters

Breaking CNN Model

from tensorflow.keras.preprocessing.image import ImageDataGenerator

import tensorflow as tf

tf.random.set\_seed(42)

train\_Datagen = ImageDataGenerator(rescale=1./255)

valid\_Datagen = ImageDataGenerator(rescale=1./255)

train\_dir = "pizza\_steak/train/"

test\_dir = "pizza\_steak/test/"

train\_data = train\_Datagen.flow\_from\_directory(train\_dir,batch\_size=32,target\_size=(224,224),class\_mode="binary",seed=42)

valid\_data = valid\_Datagen.flow\_from\_directory(test\_dir,batch\_size=32,target\_size=(224,224),class\_mode="binary",seed=42)

images , labels = train\_data.next()

len(images), len(labels)

len(train\_data)

images[:2], images[0].shape

See for paperswith code website

Inside CNN Model

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.layers import Dense, Flatten,Conv2D, MaxPool2D, Activation

from tensorflow.keras import Sequential

model\_4 = Sequential([

Conv2D(filters=10,

kernel\_size=3,

strides=1,

padding="valid",

activation="relu",

input\_shape=(224,224,3)),

Conv2D(10,3,activation="relu"),

Conv2D(10,3,activation="relu"),

Flatten(),

Dense(1,activation="sigmoid")

])

Graphical user interface, text, application

Description automatically generated

model\_4.compile(loss="binary\_crossentropy",

optimizer=Adam(),

metrics=["accuracy"])

model\_4.summary()

Fit the model

history\_4 = model\_4.fit(train\_data,

epochs=5,

steps\_per\_epoch=len(train\_data),

validation\_data=valid\_data,

validation\_steps=len(valid\_data))

Evaluating Model

import pandas as pd

pd.DataFrame(history\_4.history).plot(figsize=(10,7))

Creating evaluate function

import matplotlib.pyplot as plt

def plot\_loss\_curv(history):

loss= history.history["loss"]

val\_loss = history.history["val\_loss"]

accuracy = history.history["accuracy"]

val\_accuracy = history.history["val\_accuracy"]

epochs = range(len(history.history["loss"]))

plt.plot(epochs,loss,label="training\_loss")

plt.plot(epochs,val\_loss,label="val\_loss")

plt.title("Loss")

plt.xlabel("Epochs")

plt.legend()

plt.figure()

plt.plot(epochs,accuracy,label="training\_accuracy")

plt.plot(epochs,val\_accuracy,label="val\_accuracy")

plt.title("Accuracy")

plt.xlabel("Epochs")

plt.legend()

plot\_loss\_curv(history\_4)

Adjusting the Parameters--- Reducing over fitting also known as regularization

model\_5 = Sequential([

Conv2D(10,3,activation="relu",input\_shape=(224,224,3)),

MaxPool2D(pool\_size=2),

Conv2D(10,3,activation="relu"),

MaxPool2D(),

Conv2D(10,3,activation="relu"),

MaxPool2D(),

Flatten(),

Dense(1,activation="sigmoid")

])

model\_5.compile(loss="binary\_crossentropy",

optimizer=Adam(),

metrics=["accuracy"])

history\_5 = model\_5.fit(train\_data,

epochs=5,

steps\_per\_epoch=len(train\_data),

validation\_data=valid\_data,

validation\_steps=len(valid\_data))

Data augmentation

Data Augmentation

-process of altering our training data, leading to it having more diversity

and in turn allowing our models to learn more generalizable patterns

train\_datagen\_augmented= ImageDataGenerator(rescale=1/255.,

rotation\_range=0.2,

shear\_range=0.2,

zoom\_range=0.2,

width\_shift\_range=0.2,

height\_shift\_range=0.2,

horizontal\_flip=True)

train\_datagen = ImageDataGenerator(rescale=1/255.)

test\_datagen = ImageDataGenerator(rescale=1/255.)

print("Augmentated training images :")

train\_data\_augmented = train\_datagen\_augmented.flow\_from\_directory(train\_dir,

target\_size=(224,224),

batch\_size=32,

class\_mode="binary",

shuffle=False)

print("Non Augmentated training images :")

train\_data = train\_datagen.flow\_from\_directory(train\_dir,

target\_size=(224,224),

batch\_size=32,

class\_mode="binary",

shuffle=False)

print("Non Augmentated testing images :")

test\_data = test\_datagen.flow\_from\_directory(test\_dir,

target\_size=(224,224),

batch\_size=32,

class\_mode="binary")

images , lebels = train\_data.next()

augmented\_images , augmented\_lebels = train\_data\_augmented.next()

Compare augmented data vs original data

import random

random\_number = random.randint(0,31)

plt.imshow(images[random\_number])

print(random\_number)

plt.title("Original Image")

plt.axis(False)

plt.figure()

plt.imshow(augmented\_images[random\_number])

print(random\_number)

plt.title("Augmented Image")

plt.axis(False)

Create model with augmented datawith out shuffle

# Create the model (same as model\_5)

model\_6 = Sequential([

Conv2D(10, 3, activation='relu', input\_shape=(224, 224, 3)),

MaxPool2D(pool\_size=2), # reduce number of features by half

Conv2D(10, 3, activation='relu'),

MaxPool2D(),

Conv2D(10, 3, activation='relu'),

MaxPool2D(),

Flatten(),

Dense(1, activation='sigmoid')

])

# Compile the model

model\_6.compile(loss='binary\_crossentropy',

optimizer=Adam(),

metrics=['accuracy'])

# Fit the model

history\_6 = model\_6.fit(train\_data\_augmented, # changed to augmented training data

epochs=5,

steps\_per\_epoch=len(train\_data\_augmented),

validation\_data=test\_data,

validation\_steps=len(test\_data))

plot\_loss\_curv(history\_6)

Create model with augmented data and shuffling

train\_data\_augmented\_shuffle = train\_datagen\_augmented.flow\_from\_directory(train\_dir,

target\_size=(224,224),

batch\_size=32,

class\_mode="binary",

shuffle=True)

model\_7 = Sequential([

Conv2D(10,3,activation="relu",input\_shape=(224,224,3)),

MaxPool2D(pool\_size=2),

Conv2D(10,3,activation="relu"),

MaxPool2D(),

Conv2D(10,3,activation="relu"),

MaxPool2D(),

Flatten(),

Dense(1,activation="sigmoid")

])

model\_7.compile(loss="binary\_crossentropy",

optimizer=Adam(),

metrics=["accuracy"])

history\_7 = model\_7.fit(train\_data\_augmented\_shuffle,

epochs=5,

steps\_per\_epoch=len(train\_data\_augmented\_shuffle),

validation\_data=test\_data,

validation\_steps=len(test\_data))

plot\_loss\_curv(history\_7)

Make prediction with custom image

print(class\_names)

import matplotlib.image as mpimg

import matplotlib.pyplot as plt

!wget https://raw.githubusercontent.com/mrdbourke/tensorflow-deep-learning/main/images/03-steak.jpeg

steak = mpimg.imread("03-steak.jpeg")

plt.imshow(steak)

plt.axis(False)

Function to preprocess custom image

def load\_and\_prep\_image(filename, img\_shape=224):

img = tf.io.read\_file(filename)

img = tf.image.decode\_image(img,channels=3)

img = tf.image.resize(img,size=[img\_shape,img\_shape])

img = img/255

return img

steak = load\_and\_prep\_image("03-steak.jpeg")

steak

Expand image if shape error occur

pred = model\_7.predict(steak)

pred

ValueError: Input 0 of layer "sequential\_6" is incompatible with the layer: expected shape=(None, 224, 224, 3), found shape=(32, 224, 3)

# Add an extra axis

print(f"Shape before new dimension: {steak.shape}")

steak = tf.expand\_dims(steak, axis=0) # add an extra dimension at axis 0

#steak = steak[tf.newaxis, ...] # alternative to the above, '...' is short for 'every other dimension'

print(f"Shape after new dimension: {steak.shape}")

pred = model\_7.predict(steak)

pred

Custom Function to predict class name

def pred\_and\_plot(model,filename,class\_names= class\_names):

img = load\_and\_prep\_image(filename)

pred = model.predict(tf.expand\_dims(img,axis=0))

pred\_class = class\_names[int(tf.round(pred))]

plt.imshow(img)

plt.title(f"Prediction :{pred\_class}")

plt.axis(False)

pred\_and\_plot(model\_7,"03-steak.jpeg")

!wget https://raw.githubusercontent.com/mrdbourke/tensorflow-deep-learning/main/images/03-pizza-dad.jpeg

pred\_and\_plot(model\_7, "03-pizza-dad.jpeg", class\_names)

Multiclass classification

Become one with data

import zipfile

# Download zip file of 10\_food\_classes images

# See how this data was created - https://github.com/mrdbourke/tensorflow-deep-learning/blob/main/extras/image\_data\_modification.ipynb

!wget https://storage.googleapis.com/ztm\_tf\_course/food\_vision/10\_food\_classes\_all\_data.zip

# Unzip the downloaded file

zip\_ref = zipfile.ZipFile("10\_food\_classes\_all\_data.zip", "r")

zip\_ref.extractall()

zip\_ref.close()

import os

# Walk through 10\_food\_classes directory and list number of files

for dirpath, dirnames, filenames in os.walk("10\_food\_classes\_all\_data"):

print(f"There are {len(dirnames)} directories and {len(filenames)} images in '{dirpath}'.")

train\_dir = "10\_food\_classes\_all\_data/train/"

test\_dir = "10\_food\_classes\_all\_data/test/"

# Get the class names for our multi-class dataset

import pathlib

import numpy as np

data\_dir = pathlib.Path(train\_dir)

class\_names = np.array(sorted([item.name for item in data\_dir.glob('\*')]))

print(class\_names)

# View a random image from the training dataset

import random

img = view\_random\_image(target\_dir=train\_dir,

target\_class=random.choice(class\_names)) # get a random class name

Preprocess Data

from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Rescale the data and create data generator instances

train\_datagen = ImageDataGenerator(rescale=1/255.)

test\_datagen = ImageDataGenerator(rescale=1/255.)

# Load data in from directories and turn it into batches

train\_data = train\_datagen.flow\_from\_directory(train\_dir,

target\_size=(224, 224),

batch\_size=32,

class\_mode='categorical') # changed to categorical

test\_data = train\_datagen.flow\_from\_directory(test\_dir,

target\_size=(224, 224),

batch\_size=32,

class\_mode='categorical')

Create a model

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv2D, MaxPool2D, Flatten, Dense

# Create our model (a clone of model\_8, except to be multi-class)

model\_9 = Sequential([

Conv2D(10, 3, activation='relu', input\_shape=(224, 224, 3)),

Conv2D(10, 3, activation='relu'),

MaxPool2D(),

Conv2D(10, 3, activation='relu'),

Conv2D(10, 3, activation='relu'),

MaxPool2D(),

Flatten(),

Dense(10, activation='softmax') # changed to have 10 neurons (same as number of classes) and 'softmax' activation

])

# Compile the model

model\_9.compile(loss="categorical\_crossentropy", # changed to categorical\_crossentropy

optimizer=tf.keras.optimizers.Adam(),

metrics=["accuracy"])

# Fit the model

history\_9 = model\_9.fit(train\_data, # now 10 different classes

epochs=5,

steps\_per\_epoch=len(train\_data),

validation\_data=test\_data,

validation\_steps=len(test\_data))

# Evaluate on the test data

model\_9.evaluate(test\_data)

# Check out the model's loss curves on the 10 classes of data (note: this function comes from above in the notebook)

plot\_loss\_curves(history\_9)

Reduce overfitting by different approach

Get more data -

Simplify model -

Use data augmentation -

Use transfer learning

# Try a simplified model (removed two layers)

model\_10 = Sequential([

Conv2D(10, 3, activation='relu', input\_shape=(224, 224, 3)),

MaxPool2D(),

Conv2D(10, 3, activation='relu'),

MaxPool2D(),

Flatten(),

Dense(10, activation='softmax')

])

model\_10.compile(loss='categorical\_crossentropy',

optimizer=tf.keras.optimizers.Adam(),

metrics=['accuracy'])

history\_10 = model\_10.fit(train\_data,

epochs=5,

steps\_per\_epoch=len(train\_data),

validation\_data=test\_data,

validation\_steps=len(test\_data))

plot\_loss\_curves(history\_10)

Augmentation to reduce overfitting

# Create augmented data generator instance

train\_datagen\_augmented = ImageDataGenerator(rescale=1/255.,

rotation\_range=0.2,

width\_shift\_range=0.2,

height\_shift\_range=0.2,

zoom\_range=0.2,

horizontal\_flip=True)

train\_data\_augmented = train\_datagen\_augmented.flow\_from\_directory(train\_dir,

target\_size=(224, 224),

batch\_size=32,

class\_mode='categorical')

Clone Model Properties

# Clone the model (use the same architecture)

model\_11 = tf.keras.models.clone\_model(model\_10)

# Compile the cloned model (same setup as used for model\_10)

model\_11.compile(loss="categorical\_crossentropy",

optimizer=tf.keras.optimizers.Adam(),

metrics=["accuracy"])

# Fit the model

history\_11 = model\_11.fit(train\_data\_augmented, # use augmented data

epochs=5,

steps\_per\_epoch=len(train\_data\_augmented),

validation\_data=test\_data,

validation\_steps=len(test\_data))

# Check out our model's performance with augmented data

plot\_loss\_curves(history\_11)

**Make Prediction on custom image**

# -q is for "quiet"

!wget -q https://raw.githubusercontent.com/mrdbourke/tensorflow-deep-learning/main/images/03-pizza-dad.jpeg

!wget -q https://raw.githubusercontent.com/mrdbourke/tensorflow-deep-learning/main/images/03-steak.jpeg

!wget -q https://raw.githubusercontent.com/mrdbourke/tensorflow-deep-learning/main/images/03-hamburger.jpeg

!wget -q https://raw.githubusercontent.com/mrdbourke/tensorflow-deep-learning/main/images/03-sushi.jpeg

**Softmax function gives probablility for all classes where sigmoid function gives probility for single class**

**Adjust pred&plot function to work with multi-class**

# Adjust function to work with multi-class

def pred\_and\_plot(model, filename, class\_names):

"""

Imports an image located at filename, makes a prediction on it with

a trained model and plots the image with the predicted class as the title.

"""

# Import the target image and preprocess it

img = load\_and\_prep\_image(filename)

# Make a prediction

pred = model.predict(tf.expand\_dims(img, axis=0))

# Get the predicted class

if len(pred[0]) > 1: # check for multi-class

pred\_class = class\_names[pred.argmax()] # if more than one output, take the max

else:

pred\_class = class\_names[int(tf.round(pred)[0][0])] # if only one output, round

# Plot the image and predicted class

plt.imshow(img)

plt.title(f"Prediction: {pred\_class}")

plt.axis(False);

# Make a prediction using model\_11

pred\_and\_plot(model=model\_11,

filename="03-steak.jpeg",

class\_names=class\_names)

**Save and load model**

# Save a model

model\_11.save("saved\_trained\_model")

# Load in a model and evaluate it

loaded\_model\_11 = tf.keras.models.load\_model("saved\_trained\_model")

loaded\_model\_11.evaluate(test\_data)

# Compare our unsaved model's results (same as above)

model\_11.evaluate(test\_data)