Katherine Banis

CMPSC 497

Lab #12: Deep Learning: Transfer Learning with Tensorflow

12/12/2024

**Table of Contents**

1. Objectives (1)
2. Pseudocode/Outline (1)
3. Sample Training Images (2)
4. Steps (2)
5. Results (3)
6. Confusion Matrix (6)
7. Conclusion (6)
8. Code (7)

**Objectives:**

* To classify cookies as either “good” or “bad” using transfer learning with a pre-trained MobileNetV2 model.
* To achieve a minimum accuracy of 80% without fine-tuning.

**Pseudocode/Outline**

* Upload dataset into Colab.
* Split dataset into training and validation sets.
* Use MobileNetV2 as a base model with pre-trained weights.
* Train the model with only the top layer (frozen base layers).
* Test with real-world images and report results.

**Sample training images**

|  |  |
| --- | --- |
| good\_cookies | bad\_cookies |
| **A chocolate cookie on a white surface  Description automatically generated** | **A cookie with a bite taken out of it  Description automatically generated** |

**Explain steps in uploading and creating training/validation/testing datasets in tensorflow**

1. So I found the cookie dataset from one of the previous assignments and decided to use that. I originally tried to complete this assignment with PyCharm but was running into issues when I was trying to import keras with tensorflow. So I switched over to colab and that worked fine. I then needed to figure out how to upload the data so I made a new folder in the src folder, called it “cookies”. Then I made two new folders called “good\_cookies” and “bad\_cookies” and I uploaded the images to the correct folders.
2. When I first ran my code, without implementing step 11 which was to fine-tune the top 100 layers of MobileNetv2 to get better accuracy, I had only 50% accuracy. So I implemented step 11 only to notice no difference. I then realized that Colab had automatically created a “.ipynb\_chechpoints” folder in the “cookies” folder that was mistakenly interpreted as a class by the code. So rather than having two classes: “good\_cookies” and “bad\_cookies”, I had three. So I specified the classes in the train\_dataset and val\_dataset. .
3. That worked and my accuracy in the first run increased from 58.33% in Epoch 1 to 100% in Epoch 10. Validation accuracy also increased from 62.5% to 100%. The second run, however was very inconsistent starting as 57.08% in Epoch 1, dropping around 47% to 58%, then ending at 65% in Epoch 10. The validation accuracy remained at 100% throughout all epochs though.

**Results before fine tuning**

Epoch 1/10

4/4 ━━━━━━━━━━━━━━━━━━━━ 12s 2s/step - accuracy: 0.5833 - loss: 0.5750 - val\_accuracy: 0.6250 - val\_loss: 0.5833

Epoch 2/10

4/4 ━━━━━━━━━━━━━━━━━━━━ 8s 657ms/step - accuracy: 0.8500 - loss: 0.4723 - val\_accuracy: 0.6250 - val\_loss: 0.4968

Epoch 3/10

4/4 ━━━━━━━━━━━━━━━━━━━━ 5s 777ms/step - accuracy: 0.9083 - loss: 0.3506 - val\_accuracy: 1.0000 - val\_loss: 0.3324

Epoch 4/10

4/4 ━━━━━━━━━━━━━━━━━━━━ 5s 483ms/step - accuracy: 0.9667 - loss: 0.3112 - val\_accuracy: 0.8750 - val\_loss: 0.3113

Epoch 5/10

4/4 ━━━━━━━━━━━━━━━━━━━━ 3s 472ms/step - accuracy: 0.9792 - loss: 0.2447 - val\_accuracy: 1.0000 - val\_loss: 0.2442

Epoch 6/10

4/4 ━━━━━━━━━━━━━━━━━━━━ 7s 847ms/step - accuracy: 0.9417 - loss: 0.2189 - val\_accuracy: 1.0000 - val\_loss: 0.1756

Epoch 7/10

4/4 ━━━━━━━━━━━━━━━━━━━━ 4s 499ms/step - accuracy: 0.9667 - loss: 0.1521 - val\_accuracy: 1.0000 - val\_loss: 0.2131

Epoch 8/10

4/4 ━━━━━━━━━━━━━━━━━━━━ 5s 682ms/step - accuracy: 0.9792 - loss: 0.1322 - val\_accuracy: 1.0000 - val\_loss: 0.1581

Epoch 9/10

4/4 ━━━━━━━━━━━━━━━━━━━━ 4s 701ms/step - accuracy: 0.9792 - loss: 0.1259 - val\_accuracy: 1.0000 - val\_loss: 0.1159

Epoch 10/10

4/4 ━━━━━━━━━━━━━━━━━━━━ 5s 465ms/step - accuracy: 1.0000 - loss: 0.1104 - val\_accuracy: 1.0000 - val\_loss: 0.1199

A graph of a graph

Description automatically generated with medium confidence

**Results after fine-tuning**

Epoch 1/10

**4/4** ━━━━━━━━━━━━━━━━━━━━ **30s** 1s/step - accuracy: 0.5708 - loss: 1.6286 - val\_accuracy: 1.0000 - val\_loss: 0.1511

Epoch 2/10

**4/4** ━━━━━━━━━━━━━━━━━━━━ **5s** 993ms/step - accuracy: 0.5583 - loss: 1.6689 - val\_accuracy: 1.0000 - val\_loss: 0.1313

Epoch 3/10

**4/4** ━━━━━━━━━━━━━━━━━━━━ **6s** 845ms/step - accuracy: 0.4875 - loss: 1.6321 - val\_accuracy: 1.0000 - val\_loss: 0.1677

Epoch 4/10

**4/4** ━━━━━━━━━━━━━━━━━━━━ **5s** 821ms/step - accuracy: 0.4750 - loss: 1.5956 - val\_accuracy: 1.0000 - val\_loss: 0.1823

Epoch 5/10

**4/4** ━━━━━━━━━━━━━━━━━━━━ **6s** 1s/step - accuracy: 0.4875 - loss: 1.2541 - val\_accuracy: 1.0000 - val\_loss: 0.1901

Epoch 6/10

**4/4** ━━━━━━━━━━━━━━━━━━━━ **5s** 812ms/step - accuracy: 0.5250 - loss: 0.9770 - val\_accuracy: 1.0000 - val\_loss: 0.1500

Epoch 7/10

**4/4** ━━━━━━━━━━━━━━━━━━━━ **5s** 819ms/step - accuracy: 0.5833 - loss: 0.8581 - val\_accuracy: 1.0000 - val\_loss: 0.1517

Epoch 8/10

**4/4** ━━━━━━━━━━━━━━━━━━━━ **7s** 1s/step - accuracy: 0.5875 - loss: 0.7106 - val\_accuracy: 1.0000 - val\_loss: 0.1010

Epoch 9/10

**4/4** ━━━━━━━━━━━━━━━━━━━━ **9s** 816ms/step - accuracy: 0.5292 - loss: 0.7518 - val\_accuracy: 1.0000 - val\_loss: 0.1320

Epoch 10/10

**4/4** ━━━━━━━━━━━━━━━━━━━━ **7s** 1s/step - accuracy: 0.6500 - loss: 0.7863 - val\_accuracy: 1.0000 - val\_loss: 0.1872

A screenshot of a graph

Description automatically generated

**Confusion Matrix**

A blue square with white text

Description automatically generated

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **precision** | **recall** | **f1-score** | **support** |
| **good cookies** | 0.50 | 0.50 | 0.50 | 4 |
| **bad cookies** | 0.50 | 0.50 | 0.50 | 4 |
| **accuracy** | 0.50 | 0.50 | 0.50 | 8 |
| **macro avg** | 0.50 | 0.50 | 0.50 | 8 |
| **weighted avg** | 0.50 | 0.50 | 0.50 | 8 |

**Conclusion**

So in conclusion, I learned to specify the classes when using colab. After doing that, the transfer learning was successfully implemented using MobileNetV2 pre-trained model to classify the images of cookies. The model was first trained without fine-tuning and achieved very good results. Training accuracy and validation accuracy both reached 100% after 10 epochs.

**Code**

# step 1: import libraries

import matplotlib.pyplot as plt

import numpy as np

import os

import tensorflow as tf

from tensorflow.keras import layers, models

from tensorflow.keras.preprocessing.image import ImageDataGenerator

# import dataset

dataset\_path = "/usr/src/cookies"

# training dataset

train\_datagen = ImageDataGenerator(

rescale=1./255,

validation\_split=0.2, # 20% validation

horizontal\_flip=True,

rotation\_range=30,

)

train\_dataset = train\_datagen.flow\_from\_directory(

dataset\_path,

target\_size=(224, 224),

batch\_size=8,

subset='training',

class\_mode='binary',

classes=['good\_cookies', 'bad\_cookies']

)

val\_dataset = train\_datagen.flow\_from\_directory(

dataset\_path,

target\_size=(224, 224),

batch\_size=8,

subset='validation',

class\_mode='binary',

classes=['good\_cookies', 'bad\_cookies']

)

# test dataset

test\_dataset = val\_dataset

# set performance parameter

AUTOTUNE = tf.data.AUTOTUNE

# import pretrained MobileNetv2 CNN

# set training to 0 for all layers

# do not retrain this network, use feature extraction

base\_model = tf.keras.applications.MobileNetV2(

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

include\_top=False,

weights='imagenet',

)

base\_model.trainable = False

# add classification layer to top of network

model = models.Sequential([

base\_model,

layers.GlobalAveragePooling2D(),

layers.Dense(1, activation='sigmoid')

])

# compile and train model

model.compile(

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

loss='binary\_crossentropy',

metrics=['accuracy']

)

history = model.fit(

train\_dataset,

validation\_data=val\_dataset,

epochs=10

)

# step 10: review training statistics (accuracy and validation accuracy) and results

acc = history.history['accuracy']

val\_acc = history.history['val\_accuracy']

loss = history.history['loss']

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

epochs = range(len(acc))

plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)

plt.plot(epochs, acc, label='Training Accuracy')

plt.plot(epochs, val\_acc, label='Validation Accuracy')

plt.legend()

plt.title('Accuracy')

plt.subplot(1, 2, 2)

plt.plot(epochs, loss, label='Training Loss')

plt.plot(epochs, val\_loss, label='Validation Loss')

plt.legend()

plt.title('Loss')

plt.show()

# step 11: fine-tune the top 100 layers of MobileNetv2 to get better accuracy if needed.

# NOTE: if accuracy is below (80%, include fine-tuning in step 1)

# Unfreeze the top layers of the MobileNetV2 model for fine-tuning

base\_model.trainable = True

# Freeze all layers except the top 100

for layer in base\_model.layers[:-100]:

layer.trainable = False

# Compile the model with a lower learning rate

model.compile(

optimizer=tf.keras.optimizers.Adam(learning\_rate=1e-5),

loss='binary\_crossentropy',

metrics=['accuracy']

)

# Fine-tune the model

history\_fine = model.fit(

train\_dataset,

validation\_data=val\_dataset,

epochs=10

)

# Plot fine-tuning results

acc\_fine = history\_fine.history['accuracy']

val\_acc\_fine = history\_fine.history['val\_accuracy']

loss\_fine = history\_fine.history['loss']

val\_loss\_fine = history\_fine.history['val\_loss']

plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)

plt.plot(epochs, acc\_fine, label='Fine-Tuning Training Accuracy')

plt.plot(epochs, val\_acc\_fine, label='Fine-Tuning Validation Accuracy')

plt.legend()

plt.title('Fine-Tuning Accuracy')

plt.subplot(1, 2, 2)

plt.plot(epochs, loss\_fine, label='Fine-Tuning Training Loss')

plt.plot(epochs, val\_loss\_fine, label='Fine-Tuning Validation Loss')

plt.legend()

plt.title('Fine-Tuning Loss')

plt.show()

# Generate predictions on validation data

y\_pred = model.predict(val\_dataset)

y\_pred\_classes = (y\_pred > 0.5).astype("int32").flatten() # Convert sigmoid output to binary classes

# True labels

true\_classes = val\_dataset.classes

# Confusion matrix

cm = confusion\_matrix(true\_classes, y\_pred\_classes)

sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")

plt.xlabel('Predicted')

plt.ylabel('True')

plt.show()

# Classification report

print(classification\_report(true\_classes, y\_pred\_classes, target\_names=val\_dataset.class\_indices.keys()))