Brian Elinsky

Professor Lawrence Fulton, Ph.D.

2020SP_MSDS_422-DL_SEC55

20 May 2020

Assignment 7: Image Processing with a CNN

PROBLEM DESCRIPTION

We are providing advice to a website provider that wants to automatically classify images. The goal is the highest possible accuracy. We are OK with sacrificing training time to get better accuracy. The solution will be used to classify images that end users submit.

RESEARCH DESIGN AND MODELING METHODS

I decided to import an Xception model that was pretrained on the ImageNet dataset. I used that CNN to extract features from the labeled pictures. Then I fed those pictures into a small neural network designed to be a binary classifier. The first layer was a dense layer, the second was a dropout layer, and the third was a one-node dense layer with sigmoid activation.

For the 2x2 factorial design, 2 models had the dropout layer, 2 did not. I also varied the number of epochs, with 2 models using 30, and 2 models using 10.

DATA PREPARATION

I did very little pre-processing. I converted each image into a 3-dimensional array. I created a layer for the red, another for the green, and a third blue component of the image. Then I scaled the components to be between 0 and 1.

RESULTS AND MODEL EVALUATION

Both models with dropout outperformed the models without dropout. The effects of 10 vs 30 epochs of training were more mixed. The best model had a validation accuracy of 98.8%. The Kaggle Log Loss scores were: 0.40068, 0.35403, 0.42839, 0.42839 (Account User ID 4810027)(https://www.kaggle.com/brianelinsky/).

MANAGEMENT RECOMMENDATIONS

With accuracy being a priority, using a pre-trained CNN is a must. I would recommend moving forward with the pre-trained Xception model plus the neural network. Additional work could be done to augment the image dataset. That would increase the number of training examples, and it would make the model more robust.

```
kaggle competitions submit -c dogs-vs-cats-redux-kernels-edition -f submission.csv -m "Message"
```

3 submissions for Brian Elinsky

Sort by Most recent

All Successful Selected

Submission and Description	Private Score	Public Score	Use for Final Score
model_no_dropout_30_epochs_predictions.csv a few seconds ago by Brian Elinsky Pre-trained Xception model + train neural network from scratch No dropout layer 30 epochs of training used predict_proba instead of predict	0.42839	0.42839	☐
model_no_dropout_10_epochs_predictions.csv a minute ago by Brian Elinsky Pre-trained Xception model + train neural network from scratch No dropout layer 10 epochs of training used predict_proba instead of predict	0.42839	0.42839	☐
model_dropout_30_epochs_predictions.csv 2 minutes ago by Brian Elinsky Pre-trained Xception model + train neural network from scratch Included a dropout layer 30 epochs of training used predict_proba instead of predict	0.40068	0.40068	☐
model_dropout_10_epochs_predictions.csv 7 minutes ago by Brian Elinsky Pre-trained Xception model + train neural network from scratch Included a dropout layer 10 epochs of training used predict_proba instead of predict	0.35403	0.35403	☐

```
import os

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from keras_preprocessing.image import ImageDataGenerator
from sklearn.model_selection import train_test_split
from tensorflow.keras.applications import Xception
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.models import Sequential

# Constants
DATA_DATE = '2020-05-17'
BATCH_SIZE = 32
IMG_HEIGHT = 224
IMG_WIDTH = 224
train_dir = '/Users/brianelinsky/Dropbox/ActiveProjects/modeling_projects/catdog/data/2020-05-17/train'
test_dir = '/Users/brianelinsky/Dropbox/ActiveProjects/modeling_projects/catdog/data/2020-05-17/test1'

# Get train filenames
train_filenames = os.listdir(train_dir)

# Get test filenames
test_filenames = os.listdir(test_dir)
test_filenames = sorted(test_filenames, key=lambda x: float(x[:-4]))

# Label each training image as dog or cat
categories = []
for filename in train_filenames:
    category = filename.split('.')[0]
    if category == 'dog':
        categories.append('dog')
    else:
        categories.append('cat')

# Create training dataframe of filenames and labels
train_df = pd.DataFrame({
    'filename': train_filenames,
    'category': categories
})

# Create test dataframe for test filenames
test_df = pd.DataFrame({
    'filename': test_filenames
})

# Split training data into train and validate datasets
train_df, validate_df = train_test_split(train_df, test_size=0.15, random_state=22)

# Calculate dataset sizes
train_data_count = len(train_df)
validate_data_count = len(validate_df)
test_data_count = len(test_df)

# Load a pre-trained CNN
conv_base = Xception(weights='imagenet',
                     include_top=False,
                     input_shape=(IMG_WIDTH, IMG_HEIGHT, 3))

print(conv_base.summary())

# Create generator to rescale images by 1/255
datagen = ImageDataGenerator(rescale=1. / 255)


def extract_features(df, sample_count):
    features = np.zeros(shape=(sample_count, 7, 7, 2048))
    labels = np.zeros(shape=sample_count)
    generator = datagen.flow_from_dataframe(
        df,
        train_dir,
        x_col='filename',
        y_col='category',
```

```
        target_size=(IMG_HEIGHT, IMG_WIDTH),  # Resize images to target
        color_mode='rgb',
        class_mode='binary',
        batch_size=BATCH_SIZE
    )
    i = 0
    for inputs_batch, labels_batch in generator:
        features_batch = conv_base.predict(inputs_batch)
        features[i * BATCH_SIZE: (i + 1) * BATCH_SIZE] = features_batch
        labels[i * BATCH_SIZE: (i + 1) * BATCH_SIZE] = labels_batch
        i += 1
        if i * BATCH_SIZE >= sample_count:
            break
    return features, labels


def extract_test_features(df, sample_count):
    features = np.zeros(shape=(sample_count, 7, 7, 2048))
    generator = datagen.flow_from_dataframe(
        df,
        test_dir,
        x_col='filename',
        target_size=(IMG_HEIGHT, IMG_WIDTH),  # Resize images to target
        color_mode='rgb',
        class_mode=None,
        shuffle=False,
        batch_size=BATCH_SIZE
    )
    i = 0
    for inputs_batch in generator:
        features_batch = conv_base.predict(inputs_batch)
        features[i * BATCH_SIZE: (i + 1) * BATCH_SIZE] = features_batch
        i += 1
        if i * BATCH_SIZE >= sample_count:
            break
    return features


# Use pre-trained CNN to extract features for train and validation datasets
train_features, train_labels = extract_features(train_df, train_data_count)
validation_features, validation_labels = extract_features(validate_df, validate_data_count)

# Extract features from test dataset
test_features = extract_test_features(test_df, test_data_count)

# Flatten train, test, and validate features so they can be fed into a neural network
train_features = np.reshape(train_features, (train_data_count, 7 * 7 * 2048))
validation_features = np.reshape(validation_features, (validate_data_count, 7 * 7 * 2048))
test_features = np.reshape(test_features, (test_data_count, 7 * 7 * 2048))


# CREATE 2x2 FACTORIAL DESIGN
def create_model(hasDropout):
    model = Sequential()
    model.add(Dense(256, activation='relu', input_dim=7 * 7 * 2048))
    if hasDropout:
        model.add(Dropout(0.5))
    model.add(Dense(1, activation='sigmoid'))
    return model


model_dropout_30_epochs = create_model(True)
model_dropout_10_epochs = create_model(True)
model_no_dropout_30_epochs = create_model(False)
model_no_dropout_10_epochs = create_model(False)

models = [model_dropout_30_epochs, model_dropout_10_epochs, model_no_dropout_30_epochs, model_no_dropout_10_epochs]
for model in models:
    model.compile(optimizer='RMSprop', loss='binary_crossentropy', metrics=['acc'])


def fit_model(model, num_epochs):
    return model.fit(train_features, train_labels, epochs=num_epochs,
                     validation_data=(validation_features, validation_labels))
```

```
# Fit Models
history_dropout_30 = fit_model(model_dropout_30_epochs, 30)
history_dropout_10 = fit_model(model_dropout_10_epochs, 10)
history_no_dropout_30 = fit_model(model_no_dropout_30_epochs, 30)
history_no_dropout_10 = fit_model(model_no_dropout_10_epochs, 10)


def plot_learning_curve(history, name):
    pd.DataFrame(history.history).plot(figsize=(8, 5))
    plt.grid(True)
    plt.gca().set_ylim(0, 1)
    plt.savefig(name)


# Plot learning curves
plot_learning_curve(history_dropout_30, 'model_dropout_30_epochs')
plot_learning_curve(history_dropout_10, 'model_dropout_10_epochs')
plot_learning_curve(history_no_dropout_30, 'model_no_dropout_30_epochs')
plot_learning_curve(history_no_dropout_10, 'model_no_dropout_10_epochs')


def make_prediction(model, model_name):
    model_predictions = model.predict_proba(test_features)
    model_predictions = model_predictions.tolist()
    flatten = lambda l: [item for sublist in l for item in sublist]
    model_predictions = flatten(model_predictions)
    model_predictions_series = pd.Series(model_predictions, name="label")

    # Output predictions
    image_ids = pd.Series(data=range(1, test_data_count + 1), name="id")
    output = pd.concat([image_ids, model_predictions_series], axis=1)
    output.to_csv(model_name + "_predictions.csv", index=False)


make_prediction(model_dropout_30_epochs, 'model_dropout_30_epochs')
make_prediction(model_dropout_10_epochs, 'model_dropout_10_epochs')
make_prediction(model_no_dropout_30_epochs, 'model_no_dropout_30_epochs')
make_prediction(model_no_dropout_30_epochs, 'model_no_dropout_10_epochs')
```