KENNESAW STATE
UNIVERSITY

CS 4732
MACHINE VISION

PROJECT 4

DEEP LEARNING

INSTRUCTOR

Dr. Mahmut KARAKAYA

Michael Rizig
001008703

1. ABSTRACT

In this project, we are tasked with taking a pretrained deep learning neural net and retraining it on our own dataset. For this project, I've decided to go with AlexNet as it is a simple to set up, transfer learn, and is an effective approach to this objective. We then must retrain our CNN on the DR and nonDR for different learning rates, batch sizes, and epochs. Comparing these results will help us optimize and correct our learning and training to hopefully get better accuracy. Finally, we must create a table for our results to compare.

To view all edits, changes, and see step by step revision history, view this project on my GitHub:

https://github.com/michaelrzg/CS4732-Projects

2. TEST RESULTS

2.1 Augmentation Options

For training, we utilized many different learning rates, minibatch sizes, and max-epochs and compares their accuracy to determine which will perform the best. A table of each configuration is displayed below including its datagen arguments.

Run	*Learning Rate*	*Minibatch Size*	*Max Epochs*	*Arguments*
1	**.001**	**16**	**20**	**RandxTranslation**
2	**.001**	**32**	**25**	**RandxReflection**
3	**.001**	**32**	**20**	**none**
4	**.0001**	**16**	**20**	**RandxReflection RandyTranslation**
5	**.0001**	**64**	**30**	**none**
6	**.0005**	**32**	**20**	**none**
7	**.0005**	**64**	**25**	**RandXReflection RandXTranslation RandYTranslation**

Figure 2.1: This table displays each run and its configuration.

2.2 Accuracy

The table below shows the accuracy of each run. Because of the changing variables, we have differing accuracy from run to run:

Run	Accuracy
1	.878
2	.880
3	.875
4	.878
5	.895
6	.910
7	.902

Figure 2.2: Table displaying the accuracy of each run.

2.3 Confusion matrix for best run.

Run 6 was the best performing run of our trials. Below is the confusion matrix.

	Predicted Correctly	**Predicted Incorrectly**	**Total**
DR	**221**	**22**	**243**
Non-DR	**153**	**15**	**168**

Figure 2.3: Confusion matrix for Run 6.

2.4 Discussion

In this project, we performed transfer learning on AlexNet with our own dataset and fine tuned the parameters to maximize our accuracy for our dataset. We tried many different configuration, including tampering with the X an Y translations, reflections. Our primary options were Minibatch size, max epoch, and learning rate, with these values having the highest impact on results. As we can see from figure 2.2, our run 6, (.0005 learning rate, 32 Minibatch size, 20 max epoch, no translations) had the best results. Based on this we created a confusion matrix to show run 6's accuracy and how well it performed on both types of data. Our results show that in general, the runs with more augments / transformations performed slightly worse than runs without. We can also see that the smaller learning rates often performed on par or better than the larger rates. Given more time, it would be my intention to apply this to a different dataset, and possibly utilize this in a project with more real world applications.

3. CODE

3.1 Code for deepLearningClassifier.py

Some of the code used below came from the slides presented in class.

Credit: Lecture 12: CNN Architectures (slides 75-76)

```
# Michael Rizig
# Project 4: Deep Learing for Classification
# 001008703
# File 1: deepLearingClassifier.py
# 7/12/2024

#import Alexnet and other tools
import numpy as np
import tensorflow as tf
from keras.preprocessing.image import ImageDataGenerator
from keras.applications import AlexNet
from keras.models import Sequential
from keras.layers import Dense, Flatten, Dropout
from keras.optimizers import SGDimport
from skimage import io
import os

#Prepare Data:

# Define data generator (from Lecture 12 Slide 75)
datagen = ImageDataGenerator(
rescale=1./255,
shear_range=0.2,
zoom_range=0.2,
horizontal_flip=True,
validation_split=0.3) # Split data into training and validation
```

```
# Load and prepare data (from Lecture 12 Slide 75)
data_dir = 'Train/' # Path to your data folder
batch_size = 32
train_generator = datagen.flow_from_directory(
data_dir,
target_size=(227, 227),
batch_size=batch_size,
class_mode='categorical',
subset='training') # Use 70% of data for training
val_generator = datagen.flow_from_directory(
data_dir,
target_size=(227, 227),
batch_size=batch_size,
class_mode='categorical',
subset='validation') # Use 30% of data for validation
#create groupings for labels
nonDRLabels = []
DRLabels = []

#retreive labels from title of each image
labels = os.listdir('Train/')

#seperate id from label and pass into correct group
for s in labels:
    tag = s.split('-')[1]
    if tag == '0.jpg':
        nonDRLabels.append(s)
    elif tag == '2.jpg' or tag == '3.jpg':
        DRLabels.append(s)

print(len(nonDRLabels))
print(len(DRLabels))
#load images into memory:
#create list for image objects:
nonDRImages = []
DRImages = []

#load images into list
for s in nonDRLabels:
    nonDRImages.append(io.imread(f'Train/{s}'))

for s in DRLabels:
    DRImages.append(io.imread(f'Train/{s}'))
```

```
#ensure loading was successful:
assert len(nonDRImages) == len(nonDRLabels) and len(DRImages) == len(DRLabels)

print(nonDRImages[0].shape)

#repeat for test data:

testLabels = os.listdir('Test/')

nonDRTest = []
DRTest = []

for label in testLabels:
    tag = label.split('-')[1]
    if tag == '0.jpg':
        nonDRTest.append(io.imread(f'Test/{label}'))
    elif tag == '2.jpg' or tag == '3.jpg':
        DRTest.append(io.imread(f'Test/{label}'))



#load Alexnet (from Lecture 12 Slides 76)
pretrained = AlexNet.pretrained(weights='imagenet',include_top=False,
input_shape=(1028,1062,3))

for layer in pretrained.layers:
    layer.trainable = False

newModel = AlexNet.Sequential()

newModel.add(pretrained)

newModel.add(AlexNet.Flatten())
newModel.add(AlexNet.Dense(256, activation='relu'))
newModel.add(AlexNet.Dropout(0.5))
newModel.add(AlexNet.Dense(10, activation='softmax')) # Assuming 10 classes
# Compile the newModel
newModel.compile(optimizer=SGD(lr=0.001, momentum=0.9),
loss='categorical_crossentropy', metrics=['accuracy'])
# Train the newModel
history = newModel.fit(
AlexNet.train_generator,
steps_per_epoch=len(AlexNet.train_generator),
epochs=5,
validation_data=AlexNet.val_generator,
```

```
validation_steps=len(AlexNet.val_generator))
# Save the trained newModel
newModel.save('alexnet_transfer_learning.h5')
```