Jackson Lyons
MTH 496 Machine Learning
April 18, 2023

Final Project: Image Classification

Image classification is a machine learning problem where a set of target classes is defined, and a model is trained to recognize them using labeled example photos (1). Though not as common, image classification can also be implemented through unsupervised learning. In unsupervised image classification, raw unlabeled images are passed through the model which then identifies patterns and clusters images into groups. In this project, I will focus on supervised image classification. There are 3 main steps involved in supervised image classification:

1) Pre-processing
2) Feature Extraction
3) Model Training/Evaluation

Image pre-processing is used to prepare image data for the model. This may include techniques such as resizing, grayscale conversion, normalization and data augmentation to name a few. Pre-processing images is an important step for improving the quality of data and reducing computational complexity. (2)

Feature extraction is a crucial step in the image classification process, which involves identifying visual patterns in an image that will be used to differentiate one object from another. Feature extraction improves the performance of machine learning models by focusing on the most relevant and significant aspects of the data. Without it, models would have to analyze complete images which takes a long time and requires intensive computational resources. Some of the techniques that are practiced for feature extraction are texture analysis and edge detection. (3)

Model training involves feeding labeled data to the image classification model to learn from. The model then uses this knowledge to predict the class of new images. Once the model is trained, the model's accuracy needs to be evaluated. To do this, the model can be tested on new data (images not seen during the training process). Useful metrics for model evaluation include confusion matrix, accuracy, precision, and recall. (4)

Image classification is used today in a wide array of applications including medical imaging, autonomous driving systems, and quality control in manufacturing. Some examples of popular machine learning models for image classification are Convolutional Neural Networks (CNN), Support Vector Machines (SVM), Random Forests, and K-Nearest-Neighbors (KNN).

Application:

Another application of image classification is animal species identification. Machine learning models can be trained to classify animal species in an image. These models can be incredibly useful for wildlife biologists and ecologists studying animal populations. Conventional methods for monitoring animal populations are time-consuming and expensive (5). With recent advances in technology and an ever-increasing supply of data, machine learning and image classification specifically will likely become a bigger player in wildlife research efforts.

Isle Royale is an island located in the northwest of Lake Superior and part of the U.S. state of Michigan. Isle Royale National Park is made up of the island and the 450 nearby smaller islands and waters (6). The populations of moose and wolves on the island are closely studied. Trail cameras that are already in place on the island could be used in conjunction with image classification models to asynchronously monitor the wolf and moose populations on the island.

With this in mind, I decided to create and train four different image classification models to identify animals in images as wolves, moose, or others. The four models I used were Convolutional Neural Networks (CNN), Support Vector Machines (SVM), Random Forests,

and K-Nearest-Neighbors (KNN).

To train these models, I needed to collect image data for training. I gathered images of moose, wolves, and other animals from the web. I created a dataset containing 363 images. 121 images for each class (moose, wolf, other). Note how small this dataset is. It is common for image classification models to be trained on a dataset with 1000+ images per class.

Model Implementation:

For the implementation of SVM, KNN, and Random Forests, I used built-in functions provided by Scikit-learn. The following code creates and trains the three models (SVM, KNN, Random Forests).

```
# train on augmented data
# train Support Vector Machine classifier
svm_clf = SVC(kernel = 'linear', C = 1)
svm_clf.fit(x_train_ext, y_train_ext)

# train Random Forest classifier
rf_clf = RandomForestClassifier()
rf_clf.fit(x_train_ext, y_train_ext)

# train KNN
neigh = KNeighborsClassifier(n_neighbors=6)
neigh.fit(x_train_ext, y_train_ext)
```

Testing these models on the testing data

```
# Predict testing images
y_pred_svm = svm_clf.predict(x_test)
y_pred_rf = rf_clf.predict(x_test)
y_pred_knn = neigh.predict(x_test)

svm_score = accuracy_score(y_pred_svm, y_test)
rf_score = accuracy_score(y_pred_rf, y_test)
knn_score = accuracy_score(y_pred_knn, y_test)

print(svm_score)
print(rf_score)
print(knn_score)
```

Produces the following accuracy scores

```
0.5205479452054794
0.5068493150684932
0.410958904109589
```

For the implementation of the CNN I used built-in functions provided by Keras in TensorFlow. The following code creates and compiles the model.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Flatten,
    Dropout

# Creating a Sequential model
model = Sequential()

model.add(Conv2D(kernel_size=(3,3), filters=32, activation='tanh',
    input_shape=(224,224,3)))
model.add(MaxPooling2D(2,2))

model.add(Conv2D(filters=64,kernel_size = (3,3),activation='tanh'))
model.add(MaxPooling2D(2,2))
model.add(Dropout(0.2))

model.add(Flatten())
```

```
model.add(Dense(64,activation='relu'))
model.add(Dense(3,activation = 'softmax'))

model.compile(
              loss="sparse_categorical_crossentropy",
              metrics=['acc'],
              optimizer='adam'
             )
```

Train the CNN model

```
hist = model.fit(train,epochs=10,validation_data=val)
```

Evaluating the model on testing data

```
evaluate = model.evaluate(test)
print(evaluate)
```

Produces the following output

```
1/1 [==============================] - 0s 255ms/step - loss: 1.0978 - acc:
    0.4062
[1.097836971282959, 0.40625]
```

As indicated by the performance metrics, all four of the models performed poorly. One factor in this lacking performance is that KNN, SVM, and Random Forest were implemented straight out of the box with minimal tinkering/experimentation. Another factor is that KNN, SVM, and Random Forest are relatively simple algorithms so they won't perform the best for complicated tasks such as image classification. The biggest factor in the performance of all four models is the small dataset size. Since the dataset was so small, overfitting was a problem, especially for the CNN. With a small dataset, it is easy for the network to memorize instead of generalizing. So when tested on new data, the model is inaccurate. With much more data, better pre-processing of data, and model tuning it is likely that these models could be improved. Fortunately, there are strategies to help mitigate the negative effects of insufficient data.

Transfer Learning:

One such strategy is called transfer learning. In machine learning, transfer learning is the reuse of a pre-trained model on a new problem. In this strategy, a machine utilizes the knowledge accrued from a previous task to improve generalization for another task. Transfer learning permits the ability to effectively train a neural network with limited data. (7)

In hopes of achieving higher accuracy, I decided to train a CNN using transfer learning. Today, there are many different high-performing pre-trained models available that could be used for transfer learning. One of the most popular models for image classification is VGG16. VGG16 is a specific type of CNN which was developed by Karen Simonyan and Andrew Zisserman from the University of Oxford. VGG16 was one of the top performers in the 2014 ImageNet Large Scale Visual Recognition Challenge (ILSVRC). (8)

The following code demonstrates the implementation of VGG16 for the application problem. Keras has a built-in function for VGG16. By importing VGG16 from Keras we can load in the pre-trained model that we can then tailor for a specific task.

```
base_model = VGG16(input_shape = (224, 224, 3), include_top = False, weights
    = 'imagenet')

base_model.trainable = False
```

By setting weights = 'imagenet', we load the convolutional layers that have been pre-trained on the ImageNet dataset. The ImageNet dataset includes a plethora of different images such as animals and vehicles. The model will be able to apply the knowledge accrued from training on ImageNet, to the application problem. By setting include_top = False in the instantiation of VGG16, we remove the three fully connected layers at the end of the model.

Thus we need to add a flatten layer and fully connected (dense) layers to the end of the model in order to classify the images for the application problem.

```
from tensorflow.keras import layers, models

flatten_layer = layers.Flatten()
dense_layer_1 = layers.Dense(50, activation='relu')
dropout_layer = layers.Dropout(0.3)
dense_layer_2 = layers.Dense(20, activation='relu')
prediction_layer = layers.Dense(3, activation='softmax')


model = models.Sequential([
    base_model,
    flatten_layer,
    dense_layer_1,
    dropout_layer,
    dense_layer_2,
    dropout_layer,
    prediction_layer
])
```

Compile and train the model

```
from tensorflow.keras.callbacks import EarlyStopping

model.compile(
    optimizer='adam',
    loss="sparse_categorical_crossentropy",
    metrics=['accuracy'],
)

es1 = EarlyStopping(monitor='accuracy', mode='max', patience=3,
   restore_best_weights=True)
es2 = EarlyStopping(monitor='loss', mode='min', patience=2,
   restore_best_weights=True)

model.fit(train, epochs=14, callbacks=[es1,es2])
```

Evaluate the model on testing data

```
evaluate = model.evaluate(test)
print(evaluate)
```

Produces the following output

```
2/2 [==============================] - 4s 1s/step - loss: 29.4763 -
   accuracy: 0.6667
[29.476301193237305, 0.6666666865348816]
```

So the pre-trained VGG16 achieved an accuracy of 66.67%. This is a large improvement from the previous models.

As indicated by the performance metrics of the four models I trained from scratch vs the pre-trained VGG16, the pre-trained VGG16 performs much better. Given the very small training dataset, it is challenging for any model to learn enough information to accurately classify new images. Due to this lack of data, overfitting was a major issue. I was able to combat the overfitting a little bit with data augmentation and dropout layers, but it was still an issue. Compared to training a CNN from scratch, pre-trained models like VGG16 can perform much better with limited data, since they leverage the knowledge like features and patterns learned from the pre-training process to improve accuracy and decrease training time. One way I could've potentially increased the accuracy further for the pre-trained VGG16 would be fine-tuning. The idea of fine-tuning is to allow the last few pre-trained layers to be retrained on your dataset. Nonetheless, given the extremely small dataset, I am pleased with the accuracy from the VGG16 model. Given this substantial improvement in

performance, one might ask what makes the VGG16 model so effective?

VGG16 Architecture:

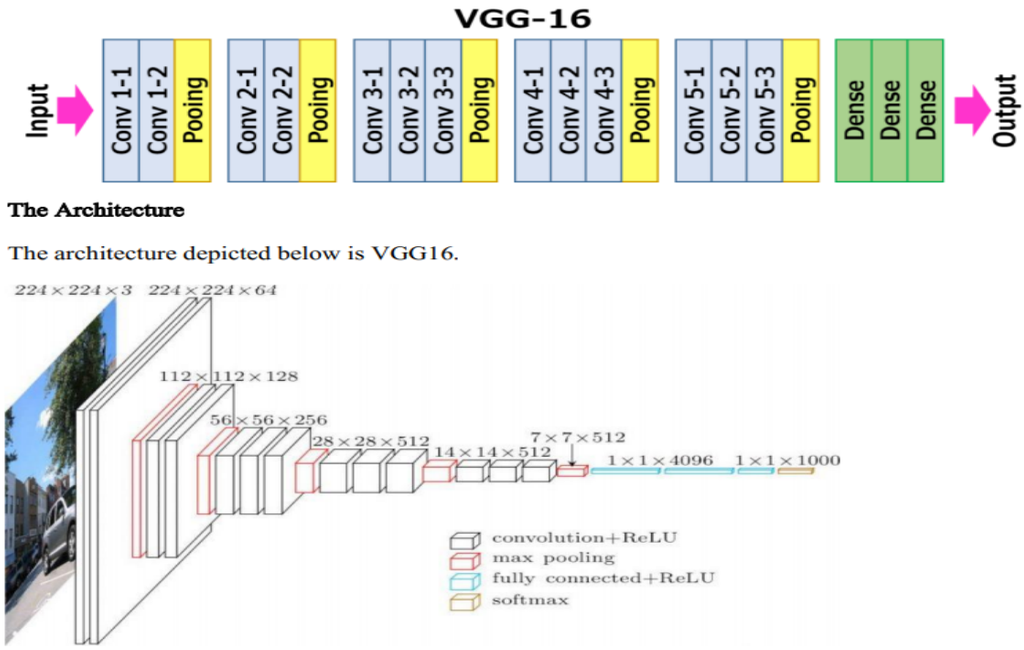

FIGURE 1. VGG16 Architecture (9)

In VGG16 there are 13 convolutional layers, 5 Max Pooling layers, and 3 Dense layers (10). Of these 21 layers, only 16 weight layers i.e., learnable parameters layer which consists of the 13 convolutional layers and 3 dense/fully connected layers. Hence the ”16” in VGG16. Compared to other high-performing models, VGG16 is unique in that its convolutional layers use fixed-size filters/kernels of the smallest possible receptive field of 3x3 (8). The idea behind using multiple convolution layers with a small fixed-size receptive field, is that stacked 3x3 filters can achieve the same effective receptive field as a single layer with a large filter. For example, 2 stacked convolution layers with filters of size 3x3 are equivalent to one convolution layer with 5x5 filter. This approach has two main benefits: 1. having multiple convolution layers with a smaller receptive field will reduce the number of parameters needed to be trained (ex. 5x5 has 25 parameters, 2(3x3) has 18 parameters) and 2. having more convolution layers means more non-linear rectification layers which makes the decision function more discriminative. This allows the CNN to converge faster (11). While this approach may decrease the number of parameters from convolution layers, VGG16 as a whole is a very large deep neural network with a total of 138 million parameters. This is due to the depth of the network and having 3 fully connected layers. For comparison, AlexNet has a total of 62 million parameters.

Conclusion:

For a human image classification is a relatively simple task. However, teaching a machine to classify images can be much more challenging. This process usually requires large amounts of labeled data and rigorous pre-processing before the model can begin to be trained. For example, the VGG16 implementation I utilized, was pre-trained on the ImageNet dataset which contains over 14 million images covering 1000 different classes. One takeaway from this project would be the difficulty of training a CNN. First a large amount of labeled data needs to be collected. Typically 1000+ images per class. Training a CNN (even with a small dataset) is time-consuming and computationally intensive. There are many different layers and parameters to tinker with that can make the process of training a CNN a daunting task. The benefit is, when done correctly CNN’s can produce highly accurate image classification models.

While I am pleased with the VGG16 accuracy of 66.67%, this is probably not good enough to put into practice. Furthermore, the model was trained on high-quality clear images, whereas in the wild, images captured by trail cameras are typically poor quality, so it is likely the

model would perform worse in practice. Nevertheless, provided an adequate amount of labeled data (specifically images generated from trail cameras), it is very possible to train a highly accurate model that could be implemented in wildlife research efforts to minimize time-consuming traditional methods of population monitoring.

References:

1) "ML Practicum: Image Classification — Machine Learning — Google for Developers." Google, n.d. https://developers.google.com/machine-learning/practica/image-classification.
2) Bhupathiraju, Subhadra. "What's Image Preprocessing Doing for Machine Learning Model..." Medium, April 22, 2023. https://medium.com/@shubavarma/whats-image-preprocessing-doing-for-machine-learning-model-4f77b21900a8.
3) "What Is Image Classification? Basics You Need to Know." SuperAnnotate, 30 May 2023, www.superannotate.com/blog/image-classification-basics.
4) Ibrahim, Mostafa. "A Gentle Introduction to Image Classification." W&B, 27 Jan. 2023, wandb.ai/mostafaibrahim17/ml-articles/reports/A-Gentle-Introduction-to-Image-Classification–VmlldzozNDI4MjQ4.
5) Tuia, D., Kellenberger, B., Beery, S. et al. "Perspectives in machine learning for wildlife conservation." Nat Commun 13, 792 (2022). https://doi.org/10.1038/s41467-022-27980-y
6) Wikipedia contributors, "Isle Royale," Wikipedia, The Free Encyclopedia, https://en.wikipedia.org/w/index.php?title=Isle_Royale&oldid=1206304596.
7) Donges, Niklas. "What Is Transfer Learning? Exploring the Popular Deep Learning Approach." Built In, 2022, builtin.com/data-science/transfer-learning.
8) "Understanding VGG16: Concepts, Architecture, and Performance." Datagen, 22 May 2023, datagen.tech/guides/computer-vision/vgg16/.
9) Thaker, Tanmay. "VGG 16 Easiest Explanation." Medium, Nerd For Tech, 13 Aug. 2021, medium.com/nerd-for-tech/vgg-16-easiest-explanation-12453b599526.
10) G, Rohini. "Everything You Need to Know about VGG16." Medium, Medium, 23 Sept. 2021, medium.com/@mygreatlearning/everything-you-need-to-know-about-vgg16-7315defb5918.
11) "Introduction to VGG16: What Is VGG16?" Great Learning Blog: Free Resources What Matters to Shape Your Career!, 18 Nov. 2022, www.mygreatlearning.com/blog/introduction-to-vgg16/.