GALGOTIAS UNIVERSITY

(Established under Galgotias University Uttar Pradesh Act No. 14 of 2011)

IMAGE CLASSIFICATION BETWEEN CNN AND TRANSFER LEARNING

A Report for the evaluation 3 of project 2

Submitted by

Beenish Fazal
(1613101221/16SCSE101361)

in partial fulfillment for the award of the degree

of

Bachelor of Technology

In

Computer Science and Engineering

SCHOOL OF COMPUTING SCIENCE AND ENGINEERING

Under the Supervision of

Dr. K.M. Baalamurugan, Asst. Professor

TABLE OF CONTENTS

Abstract

CNN and Transfer Learning model algorithms are used to solve specific tasks. These are designed to work in isolation, they are used in image classification. Image classification is one of the areas where Deep Learning models are successfully applied to practical application. It is an active area of research where many approaches have been proposed and many are popping up. The purpose of this research paper is to show the difference between the classification of these two models, which model is more efficient and less time consuming. In the conclusion section of the research paper we have described which of the two algorithms for image classification is better in terms of efficiency and time taken.

Humans have a unique ability to transfer knowledge to one another. What we acquire as knowledge while learning about a task, we use in the same way to solve related tasks. The more related the tasks, the easier it is for us to transfer, or use our knowledge. Some simple examples would be. Know how to ride a bike -learn how to ride a car. Know how to play classic piano – learn how to play jazz piano. Know math and statistics - learn machine learning. In each of the above situations, we don't learn everything from the beginning, we just use the knowledge acquired from previous tasks. We transfer and use our knowledge from what we have learnt in the past!

Conventional machine learning and deep learning algorithms are so far have been tradition design to work in confinement. These algorithms are trained to solve specific problems. The models have to be built from scratch once the feature-description changes. Transfer learning is the idea of overcoming the learning and using the knowledge obtained from one task to solve related ones. In this article, we will do a comprehensive coverage of the concepts,

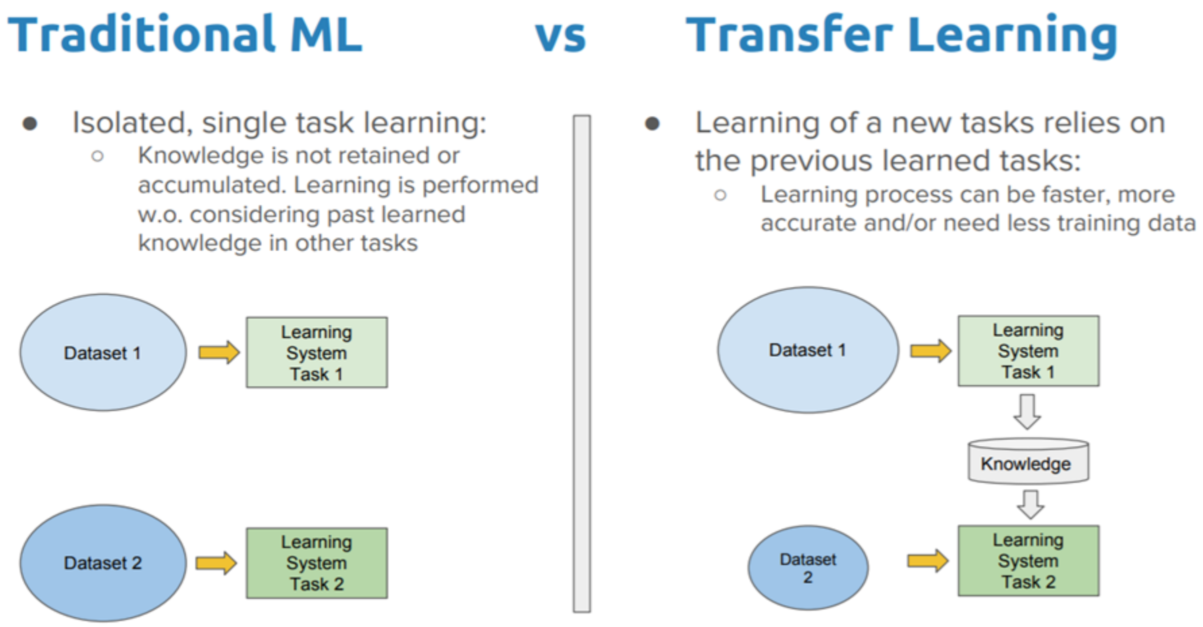
Introduction

Thus, we can use a network trained on unrelated categories in a massive dataset(usually Imagenet) and apply it to our own problem because there are universal, low-level features shared between images. The images in the Kaggle dataset are very similar to those in the imagenet dateset and the knowledge a model learns on Imagenet should easily transfer to this task.

We have already briefly discussed that humans don't learn everything from the ground up and leverage and transfer their knowledge from previously learnt domains to newer domains and tasks. Given the craze for True Artificial Gerneral Intelligence transfer learning is something which data scientists and researchers believe can further our progress towards AGI.

In fact, transfer learning is not a concept which just cropped up in the 2010s. The Neural Information Processing Systems (NIPS) 1995 workshop Learning to Learn: Knowledge Consolidation and Transfer in Inductive Systems is believed to have provided the initial motivation for research in this field. Since then, terms such as Learning to Learn, Knowledge Consolidation, and Inductive Transfer have been used interchangeably with transfer learning. Invariably, different researchers and academic texts provide definitions from different contexts. In their famous book, Deep Learning, Goodfellow et al refer to transfer learning in the context of generalization.

However, getting such a dataset for every domain is tough. Besides, most deep learning models are very specialized to a particular domain or even a specific task. While these might be state-of-the-art models, with really high accuracy and beating all benchmarks, it would be only on very specific datasets and end up suffering a significant loss in performance when used in a new task which might still be similar to the one it was trained on. This forms the motivation for transfer learning, which goes beyond specific tasks and domains, and tries to see how to leverage knowledge from pre-trained models and use it to solve new problems!


Traditional ML
vs
Transfer Learning
Isolated, single task learning:
Knowledge is not retained or accumulated. Learning is performed w.o. considering past learned knowledge in other tasks
Dataset 1
Learning System Task 1
Dataset 2
Learning System Task 2
Learning of a new tasks relies on the previous learned tasks:
Learning process can be faster, more accurate and/or need less training data
Dataset 1
Learning System Task 1
Knowledge
Dataset 2
Learning System Task 2

Traditional learning is isolated and occurs purely based on specific tasks, datasets and training separate isolated models on them. No knowledge is retained which can be transferred from one model to another. In transfer learning, you can leverage knowledge (features, weights etc) from previously trained models for training newer models and even tackle problems like having less data for the newer task!

Our task will be to train a convolutional neural network (CNN) that can identify objects in images. We'll be using the kaggle which has images in 101 categories. Most categories only have 50 images which typically isn't enough for a neural network to learn to high accuracy. Therefore, instead of building and training a CNN from scratch, we'll use a pre-built and pre-trained model applying transfer learning.

The basic premise of transfer learning is simple: take a model trained on a large dataset and transfer its knowledge to a smaller dataset. For object recognition with a CNN, we freeze the early convolutional layers of the network and only train the last few layers which make a prediction. The idea is the convolutional layers extract general, low-level features that are applicable across images — such as edges, patterns, gradients — and the later layers identify specific features within an image such as eyes or wheels.

Proposed System

Image Classification is one of the areas where Deep learning models are very successfully applied to practical applications. It is an area of research where many approaches have been proposed and many more are coming up. The most successful Deep Learning models like ImageNet, GoogleNet which perform comparison better than humans are very large and complex models.

Major components of my setup would be:

Dataset: Cats Vs Dogs dataset from Kaggle. Identifying cats in an image is a classical problem of Deep Learning. So, this dataset provides a good starting point. It has 8000 training images, 4000 images each of cats and dogs and 2000 test images, 1000 images each of cats and dogs.

Framework: Tensorflow 2.0.0 and Keras, Keras is now included in Tensorflow 2.0 so you may not need to import it separately.

CNN Model

Convolution layers have proved to be very successful in tasks involving images e.g. image classification, object identification, face recognition etc. They allow parameter sharing which results in a very optimized network compared to using Dense layers. Following is a good source of understanding Convolution Neural Networks: http://cs231n.github.io/convolutional-networks/

Image classification using CNN

The CNN model designed for this experiment has following definition:

```
model = Sequential()
model.add(Conv2D(32,(3,3), input_shape = input_shape))
model.add(Activation(‘relu’))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Conv2D(32,(3,3)))
model.add(Activation(‘relu’))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Conv2D(64,(3,3)))
model.add(Activation(‘relu’))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Flatten()) #flatter feature tensor to 1D
model.add(Dense(64))
model.add(Activation(‘relu’))
model.add(Dropout(0.5))
model.add(Dense(1))
model.add(Activation(‘sigmoid’))
model.compile(loss = ‘binary_crossentropy’,
optimizer = ‘rmsprop’,
metric = [‘accuracy’])
```

Result:

Training time: 6.4 hrs, loss: 0.0546, val_loss: 3.2969

As the results suggest, CNN works much better when dealing with images. We have reduced training time by almost 1 hr. Training loss is very optimized but validation loss is still a bit higher which indicates over-fitting. We can further regularize our model to reduce overfitting or we can use any of the methods suggested in the DNN section to improve our model performance. But,
again this is not what we will do here, instead we will directly move to our last approach which We will be using it here.

Transfer Learning Model

Transfer learning is a method of reusing the already acquired knowledge. The idea is to use a state of the art model which is already trained on a larger dataset for long time and proven to work well in related tasks. Lot of such models are available for us to use.

Keras provides these pretrained, state of the art models. Details of these models can be found at https://keras.io/applications/

We can use these models in two ways:

Direct application: In this approach,

a. We study the model to check if it can solve our target problem.

b. If yes, we need to preprocess our input according to model and then feed it to model to get results.

2. Representation Learning: In this approach, we assess that the pretrained model may not be directly applicable to our problem. But, we can use it to get a useful representation of our input data.

a. We feed input data to a pretrained model to get a representation of our data.

b. We design our own model and feed it with representations given by pre-trained models to get results.

For the task of image classification here, the 2nd approach described above is applied. Following are the typical steps of the same:

Model used: vgg16 model, https://keras.io/applications/#vgg16

Using vgg16 we get useful representation.

The representation we get from vgg16 are fed to a sequential model.

Transfer Learning for image classification[2nd Approach]

The model definition is as follows:

VGG16 model:

```
model = applications.VGG16(include_top = False, weights = ‘imagenet’)
```

Top model:

```
model = Sequential()
model.add(Flatten(input_shape = train_data.shape[1:]))
model.add(Dense(256,activation = ‘relu’))
model.add(Dropout(0.5))
model.add(Dense(1,activation = ‘sigmoid’))
model.compile(optimizer = ‘rmsprop’,
loss = ‘binary_crossentropy’,
metrics = [‘accuracy’])
```

IMPLEMENTATION

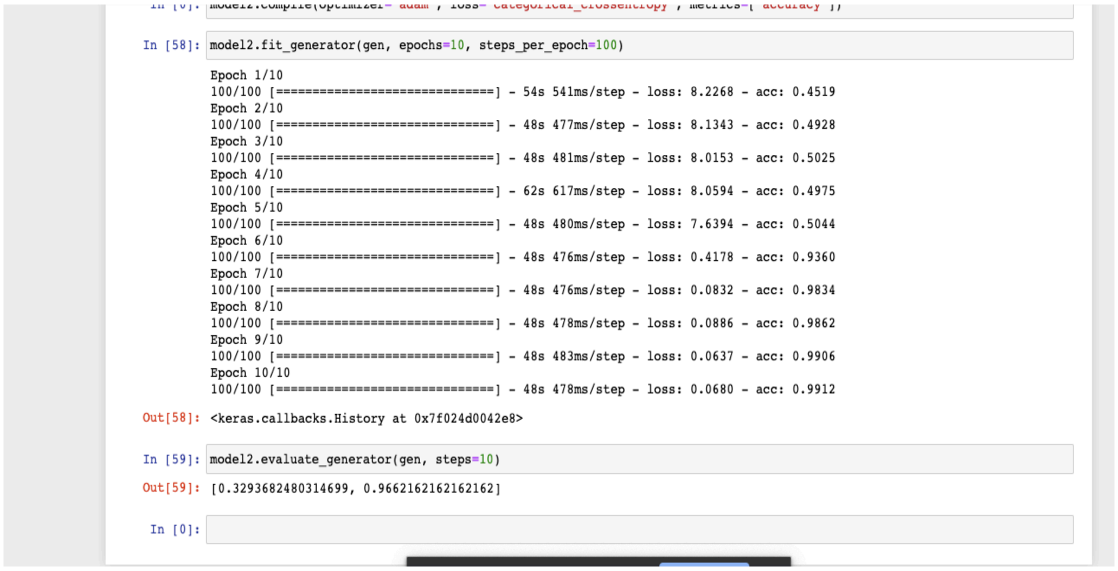
We will use the images from Kaggle to distinguish photos from one another.Here we compare images by CNN model and then transfer learning model.
Here's the code:

```
kaggle
cp kaggle.json /root/.kaggle/kaggle.json
chmod 600 /root/.kaggle/kaggle.json
kaggle datasets download ellenyusa/4classimages
unzip -qq 4classimages.zip

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from keras.preprocessing.image import ImageDataGenerator

from keras.layers import Input, Dense, MaxPool2D, Conv2D, Dropout, Flatten

from keras.models import Model

generator = ImageDataGenerator(horizontal_flip=True, zoom_range=.2, rotation_range=20,

shear_range=.3, validation_split=0.2)

train_gen = generator.flow_from_directory("data", color_mode="grayscale", subset="training")

test_gen = generator.flow_from_directory("data", color_mode="grayscale", subset="validation")

train_gen.next()[0].shape

train_gen.class_indices

images, values = train_gen.next()

images.shape

for image in images:

  img = image.reshape(256,256)

  plt.figure()

  plt.imshow(img, cmap="gray")

in_layer = Input(shape=[256, 256, 1])

layer_1 = Conv2D(32, (3,3), activation="relu")(in_layer)

layer_2 = MaxPool2D()(layer_1)

layer_3 = Conv2D(64, (3,3), activation="relu")(layer_2)

layer_4 = MaxPool2D()(layer_3)

drop = Dropout(.25)(layer_4)
```

```
layer_5 = Conv2D(128, (3,3), activation="relu")(drop)
layer_6 = MaxPool2D()(layer_5)
layer_7 = Conv2D(128, (3,3), activation="relu")(layer_6)
layer_8 = MaxPool2D()(layer_7)
layer_9 = Conv2D(128, (3,3), activation="relu")(layer_8)
layer_10 = MaxPool2D()(layer_9)
x = Conv2D(256, (3,3), activation="relu")(layer_10)
x = MaxPool2D()(x)
flatten = Flatten()(x)
d = Dense(1000, activation="relu")(flatten)
d2 = Dense(800, activation="relu")(d)
d3 = Dense(600, activation="relu")(d2)
d1 = Dense(400, activation="relu")(d3)
d2 = Dense(100, activation="relu")(d1)
d3 = Dense(50, activation="relu")(d2)
out = Dense(4, activation="softmax")(d3)
model = Model(inputs=[in_layer], outputs=[out])
model.summary()
model.compile(optimizer="adam", loss="categorical_crossentropy", metrics=["accuracy"])
hist = model.fit_generator(train_gen, epochs=10, steps_per_epoch=100,
validation_data=test_gen, validation_steps=100
plt.plot(hist.history["acc"])
from keras.applications.resnet50 import ResNet50
gen = generator.flow_from_directory("data")
base_model = ResNet50(weights="imagenet", include_top=False, input_shape=(256,256,3),
classes=len(train_gen.class_indices))
for layer in base_model.layers:
  layer.trainable = False
base_model = ResNet50(weights="imagenet", include_top=False, input_shape=(256,256,3),
classes=len(train_gen.class_indices))
```

```
for layer in base_model.layers:
  layer.trainable = False
model2 = Model(inputs=[base_model.input], outputs=[final])
model2.summary()
model2.compile(optimizer="adam", loss="categorical_crossentropy", metrics=["accuracy"])
model2.fit_generator(gen, epochs=10, steps_per_epoch=100)
model2.evaluate_generator(gen, steps=10)
```

We will have the difference between the two model accuracy as-



```
In [0]: model2.compile(optimizer="adam", loss="categorical_crossentropy", metrics=["accuracy"])

In [58]: model2.fit_generator(gen, epochs=10, steps_per_epoch=100)

         Epoch 1/10
         100/100 [==============================] - 54s 541ms/step - loss: 8.2268 - acc: 0.4519
         Epoch 2/10
         100/100 [==============================] - 48s 477ms/step - loss: 8.1343 - acc: 0.4928
         Epoch 3/10
         100/100 [==============================] - 48s 481ms/step - loss: 8.0153 - acc: 0.5025
         Epoch 4/10
         100/100 [==============================] - 62s 617ms/step - loss: 8.0594 - acc: 0.4975
         Epoch 5/10
         100/100 [==============================] - 48s 480ms/step - loss: 7.6394 - acc: 0.5044
         Epoch 6/10
         100/100 [==============================] - 48s 476ms/step - loss: 0.4178 - acc: 0.9360
         Epoch 7/10
         100/100 [==============================] - 48s 476ms/step - loss: 0.0832 - acc: 0.9834
         Epoch 8/10
         100/100 [==============================] - 48s 478ms/step - loss: 0.0886 - acc: 0.9862
         Epoch 9/10
         100/100 [==============================] - 48s 483ms/step - loss: 0.0637 - acc: 0.9906
         Epoch 10/10
         100/100 [==============================] - 48s 478ms/step - loss: 0.0680 - acc: 0.9912

Out[58]: <keras.callbacks.History at 0x7f024d0042e8>

In [59]: model2.evaluate_generator(gen, steps=10)

Out[59]: [0.3293682480314699, 0.9662162162162162]

In [0]:
```

We can clearly observe that transfer learning has 96% accuracy while CNN has only 32%.This shows that transfer learning is better than CNN inimage classification.

Conclusion

As we have seen from the proposed model that transfer learning algorithm perform better than CNN in terms of both efficiency and time so we come to a point that transfer learning is better.We know that in transfer learning we don't have to start everything from scratch it uses pre trained dataset for related problems that saves us time to start everything from scratch.

Results:

Training time: 11mins[10 mins to get representations]+1 mins to train top model.

loss: 8.2268 — accuracy: 0.4519— val_loss: 0.0680— val_accuracy: 0.9912

So, using transfer learning we could train a model which has training accuracy of 96% in only 11 mins which is much better when compared to earlier discussed models.

References

https://github.com/lalitkpal/ImageClassification.git

https://medium.com/analytics-vidhya/image-classification-a-comparison-of-dnn-cnn-and-transfer-learning-approach-704535beca25

https://towardsdatascience.com/a-comprehensive-hands-on-guide-to-transfer-learning-with-real-world-applications-in-deep-learning-212bf3b2f27a