Wildfire Detection with Deep Convolutional Neural Network by Transfer Learning

Sophia Hart, M.Sc., M.Eng.

sophia.t.hart@gmail.com

A project for my Data Science Certificate at UC Berkeley Extension

Agenda

Problem and Goal

Computer Vision using Transfer Learning

Review Jupyter Notebook

Results

Conclusion

Future Work

Q&A

Problem

Increase of CO accumulation→ Climate Change:

Temperature rise

Wildfires

Drought

Less snow

Intense hurricanes

Arctic ice shrinkage

Sea level rise

Animals endangered ...

Goal

Wildfires Detection Technologies:

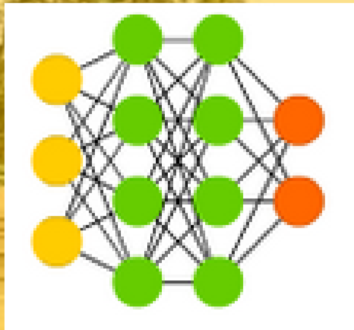
Terrestrial camera systems
Satellites
Drones
IoT

Many involve images

Goal: Train model for computer vision using deep learning

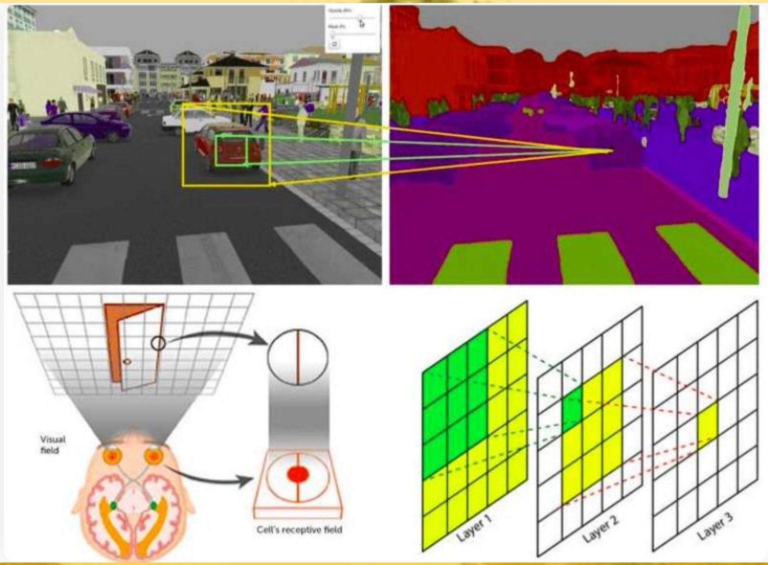
Fundamental of Convolutional Neural Networks (CNN)

Neural network with fully connected layers

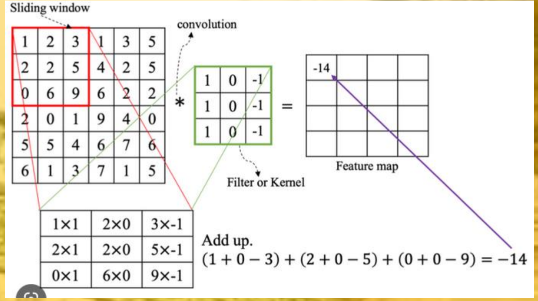


CNN utilizes receptive field

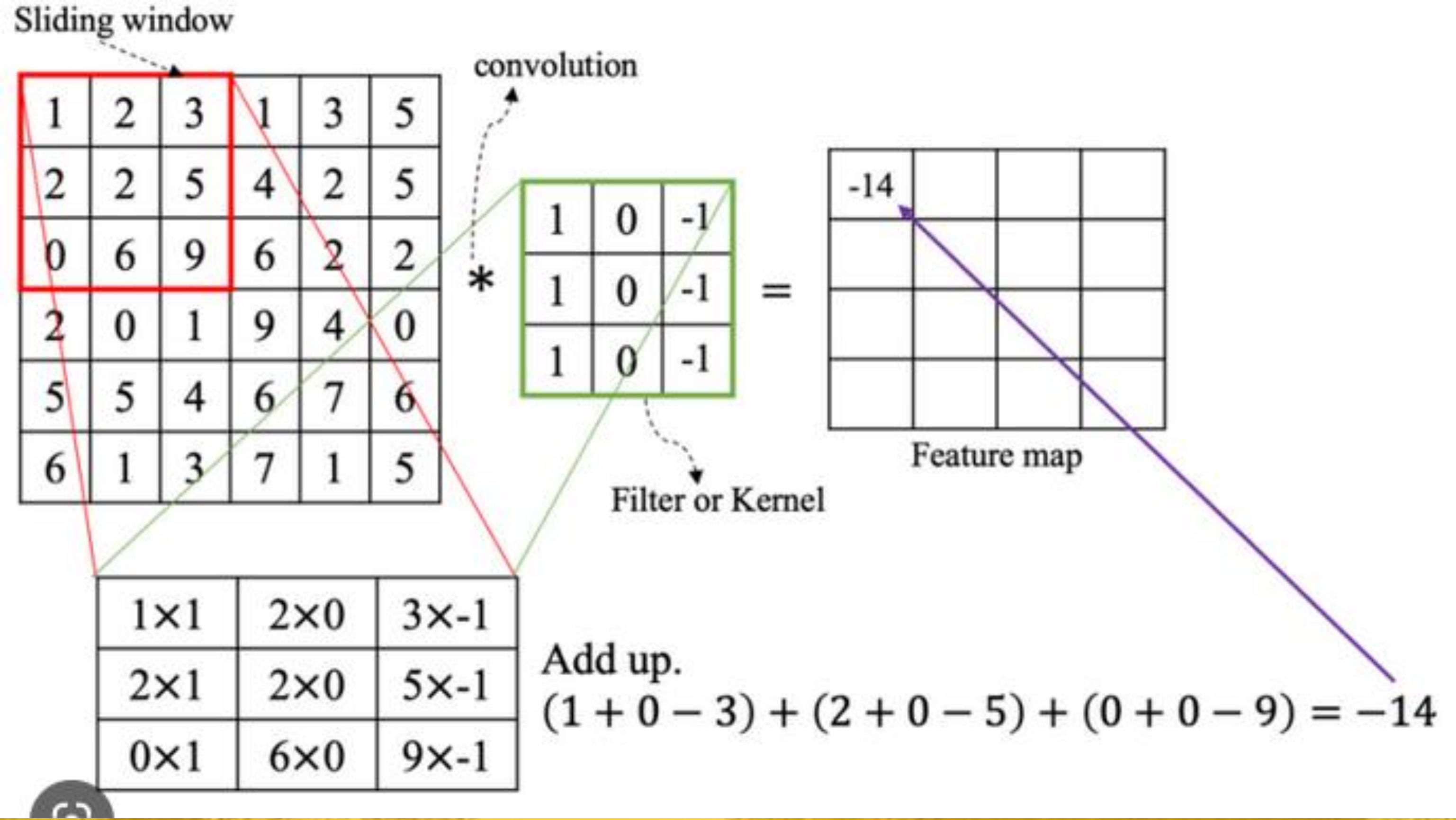
Neuron not connected to every neuron from previous layer



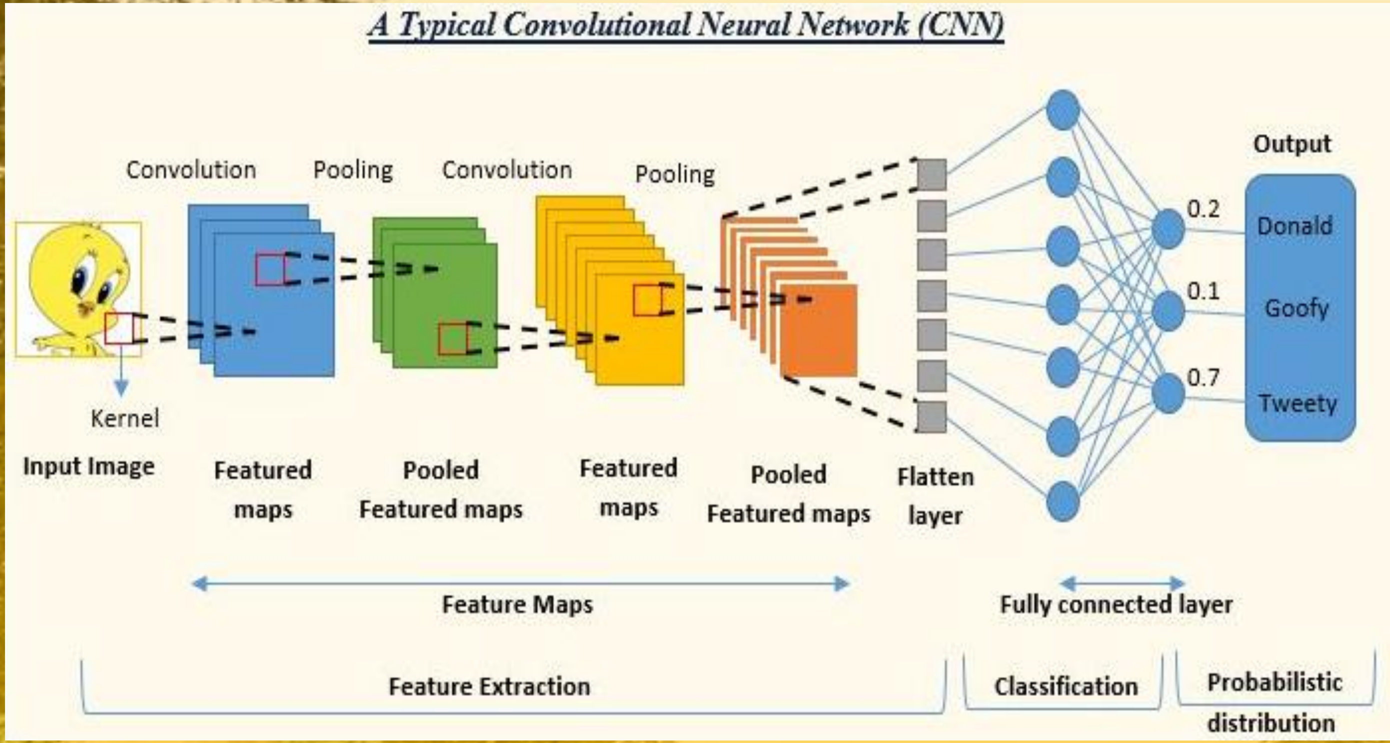
Convolution

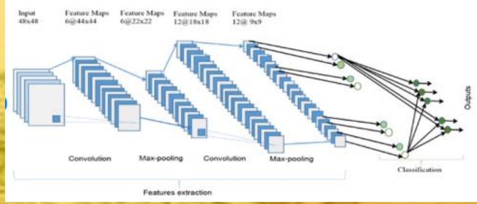


Kernel for CNN



CNN Architecture





Goal: Learn the weights/parameters for these kernels in the convolution layers.

Computer Vision using Transfer Learning

Not necessary to reinvent the wheel.
Improved learning of a new task from a related task that’s been learned.

If there’s not enough training data, good idea to reuse lower layers of pretrained model.

ImageNet Large Scale Visual Recognition Challenge (ILSVRC)

>15 million labeled images, 1000 category of objects
Took 62,000 GPU hours to train (?)

Question: Is model trained for classifying objects applicable for classifying wildfire?

Computer Vision using Transfer Learning

Keras has access to Pretrained models

Pretrained models types:
- AlexNet
- VGG
- Inception (Google)
- ResNet (Microsoft)
- EfficientNet (Google)

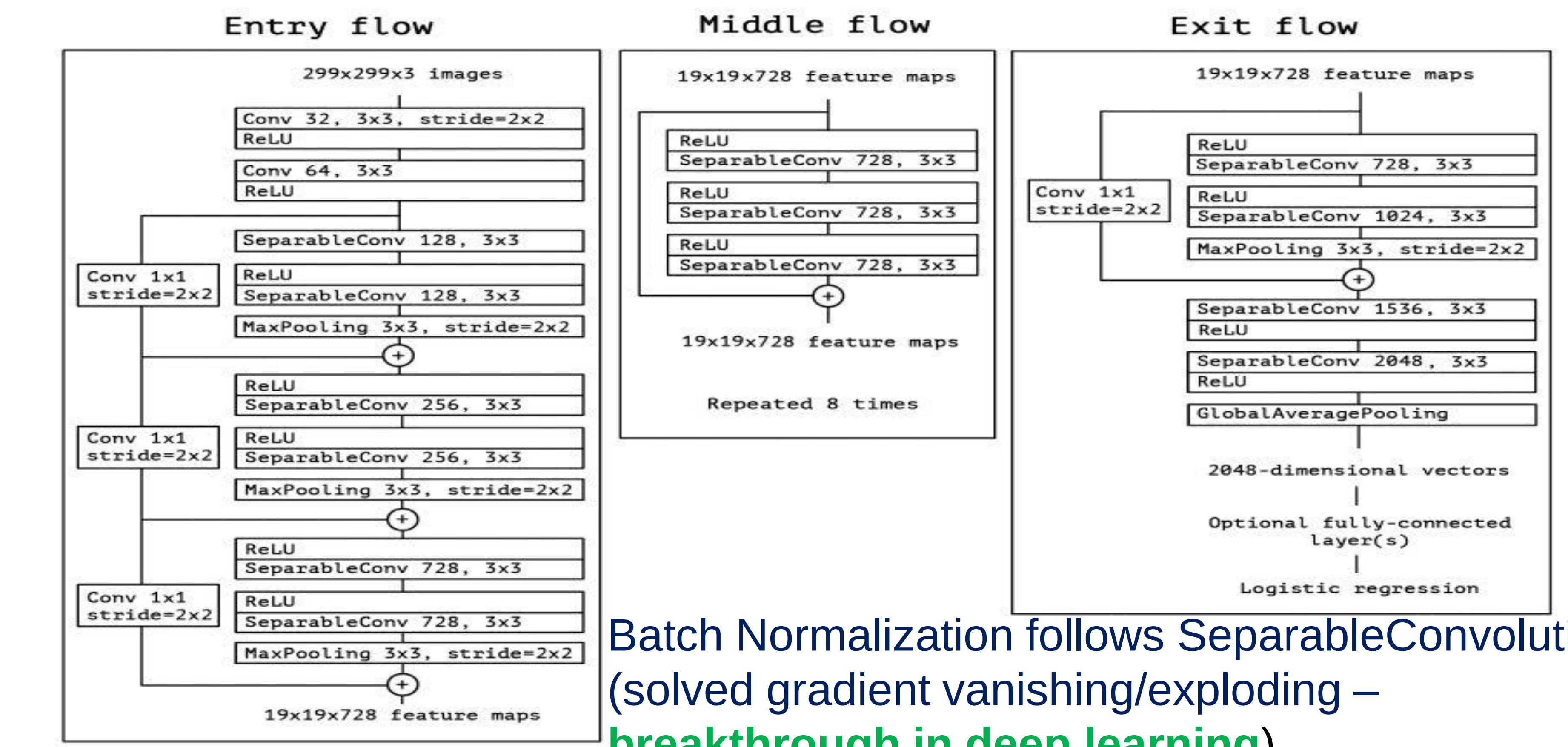
Comparison					
Network	Year	Salient Feature	top5 accuracy	Parameters	FLOP
AlexNet	2012	Deeper	84.70%	62M	1.5B
VGGNet	2014	Fixed-size kernels	92.30%	138M	19.6B
Inception	2014	Wider - Parallel kernels	93.30%	6.4M	2B
ResNet-152	2015	Shortcut connections	95.51%	60.3M	11B

My base model - Xception (Extreme Inception)

Proposed in 2016 by François Chollet, author of Keras

Less memory, exceptional accuracy

Xception Architecture



François Chollet

Batch Normalization follows SeparableConvolution (solved gradient vanishing/exploding – **breakthrough in deep learning**)

Dataset

Forest images
50 no fire
50 with fire

Data Augmentation (flips) 400 images total

Keep 3 channels (color important feature of fire)

Examples of Fire and No-fire Images

Fire

No fire

Challenge of Computer Vision for Wildfire

Fire | **No-fire**

Both flame and sunset or Autumn leaves appear orange-red

Smoke and cloud appear similar

Review Jupyter Notebook

jupyter SophiaHart_TFproj_Wildfire Last Checkpoint: 10/12/2022 (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help — Not Trusted — Python 3

Run — Markdown

In [2]:

```python
import tensorflow as tf
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from tensorflow import keras
import glob
import random

print("TensorFlow version: ", tf.__version__)

%load_ext tensorboard
```

```
2022-05-12 13:36:35.787701: W tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load dynamic l
ibrary 'libcudart.so.11.0'; dlerror: libcudart.so.11.0: cannot open shared object file: No such file or directory
2022-05-12 13:36:35.787718: I tensorflow/stream_executor/cuda/cudart_stub.cc:29] Ignore above cudart dlerror if you
do not have a GPU set up on your machine.
```

```
TensorFlow version:  2.8.0
```

Divide all jpeg files into training and testing filenames

In [3]:

```python
# Gather all the files from the fire and nonfire directories
# Some file names are jpg and some are jpeg
paths = glob.glob("./*fire/*.jp*g")

# Shuffle them because top is fire and bottom is nonfire images
random.shuffle(paths)

print("Total number of images ", len(paths))
```

```
Total number of images  100
```

In [4]:

```python
# filenames for train and test sets

train_filenames = []
test_filenames = []
i = 0

for filename in paths:

     # Send 25% (1 out of 4) of the images to testing set, the rest to training set
    if i % 4 == 0:
        test_filenames.append(filename)
    else: train_filenames.append(filename)

    i += 1

print("Original number of files for training set: ", len(train_filenames), train_filenames)
print("Original number of files for testing set: ", len(test_filenames), test_filenames)
```

```
Original number of files for training set:  75 ['./fire/fire9.jpg', './nonfire/nonfire32.jpg', './fire/fire1.jpg',
'./nonfire/nonfire38.jpg', './nonfire/nonfire51.jpg', './fire/fire14.jpg', './fire/fire13.jpg', './nonfire/nonfire2
```

Review Jupyter Notebook

jupyter SophiaHart_TFproj_Wildfire Last Checkpoint: 10/12/2022 (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Not Trusted | Python 3

Run | Markdown

Create Training and Testing Datasets

In [5]:

```
# Define function to create training and testing dataset
# data_filenames is train_filenames or test_filenames.

def create_dataset(data_filenames):

    # Create empty list for images and target (ie label: 1 is fire, 0 is nonfire)
    # The lists will be converted into tensor at the end
    image_dataset = []
    target = []
    a = 0     # index for plotting subplot
    axes = []
    rows = 10
    cols = 10
    fig = plt.figure(figsize = (50, 50))
    ax = plt.gca()

    for filename in data_filenames:
        # Read each image file
        image_file = tf.io.read_file(filename)

        # try/catch if a file is not jpeg
        try:
            image = tf.image.decode_jpeg(image_file)
        except:
            print("Not a jpeg file: ", image_file)

        # Split filename by "/". The category of fire or nonfire is the second element on the filename
        category = filename.split("/")[1]
        if category == "fire":
            label = 1
        else: label = 0

        # Resize image, Xception model expect size 224x224 images
        # Can't do resize_with_crop_or_pad, not compatible with Xception preprocess_input()
        #resize_image = tf.image.resize_with_crop_or_pad(image, 224, 224)
        resize_image = tf.image.resize(image, [224, 224])

        # Xception model needs run data through preprocess_input function
        final_image = keras.applications.xception.preprocess_input(resize_image)

        # Write image and label to datset
        image_dataset.append(np.array(final_image))
        target.append(label)

        # Display image with title (fire or nonfire category that's associated)
        axes.append(fig.add_subplot(rows,cols, a+1))
        plt.imshow(resize_image/255)
        plt.title(category, fontsize=50)
        plt.setp(plt.gcf().get_axes(), xticks=[], yticks=[]) # get rid of ticks on x-axis and y-axis
        a += 1
```

Review Jupyter Notebook

jupyter SophiaHart_TFproj_Wildfire Last Checkpoint: 10/12/2022 (autosaved)

Logou

File Edit View Insert Cell Kernel Widgets Help

Not Trusted | Python 3

Run Markdown

```python
        a += 1


        # Below is different data augmentation to increase dataset counts

        # flip left-right
        image_flipLR = tf.image.flip_left_right(resize_image)
        final_image = keras.applications.xception.preprocess_input(image_flipLR)
        # Write image and label to datset, still the same label
        image_dataset.append(np.array(final_image))
        target.append(label)

        # flipt up-down
        image_flipUD = tf.image.flip_up_down(resize_image)
        final_image = keras.applications.xception.preprocess_input(image_flipUD)
        image_dataset.append(np.array(final_image))
        target.append(label)

        # flipt left-right then up-down
        image_flipLRUD = tf.image.flip_up_down(image_flipLR)
        final_image = keras.applications.xception.preprocess_input(image_flipLRUD)
        image_dataset.append(np.array(final_image))
        target.append(label)

    return image_dataset, target

# Call the create_dataset() function to create train and test datasets
train_image_dataset, train_target = create_dataset(train_filenames)

test_image_dataset, test_target = create_dataset(test_filenames)

# Convert train_data from a list to a tensor
train_image_dataset = tf.convert_to_tensor(train_image_dataset, dtype=tf.float32)
print('Training image dataset shape: ', train_image_dataset.shape)

# Do the same for train_target
train_target = tf.convert_to_tensor(train_target, dtype=tf.uint8)

# Convert test_data from a list to tensor
test_image_dataset = tf.convert_to_tensor(test_image_dataset, dtype=tf.float32)
print('Testing image dataset shape: ', test_image_dataset.shape)

# Do the same for test_target
test_target = tf.convert_to_tensor(test_target, dtype=tf.uint8)
```

Training Dataset (part 1 of 3)

```
Training image dataset shape:  (300, 224, 224, 3)
Testing image dataset shape:  (100, 224, 224, 3)
```

Training Dataset (part 2 of 3)

Training Dataset (part 3 of 3)

Test Dataset

Review Jupyter Notebook

jupyter SophiaHart_TFproj_Wildfire Last Checkpoint: 10/12/2022 (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Not Trusted Python 3

Run Code

Save datasets into files

In [6]:

```
# Save datasets into a file for later usage

np.save('train_image_dataset.npy', train_image_dataset)
np.save('train_target.npy', train_target)
np.save('test_image_dataset.npy', test_image_dataset)
np.save('test_target.npy', test_target)
```

Load datasets from files

In [7]:

```
# Load datasets from saved files

train_image_dataset = np.load("train_image_dataset.npy")
print("Shape of train data: ", train_image_dataset.shape)
print(type(train_image_dataset))

test_image_dataset = np.load("test_image_dataset.npy")
print("Shape of test data: ", test_image_dataset.shape)

train_target = np.load("train_target.npy")
print("Shape of train target", train_target.shape)

test_target = np.load("test_target.npy")
print("Shape of test target", test_target.shape)

# Convert train_data from a list to a tensor
train_image_dataset = tf.convert_to_tensor(train_image_dataset, dtype=tf.float32)
print('Training image dataset shape: ', train_image_dataset.shape)
print(type(train_image_dataset))
print(train_image_dataset.dtype)

# Do the same for train_target
train_target = tf.convert_to_tensor(train_target, dtype=tf.uint8)
print(type(train_target))

# Convert test_data from a list to tensor
test_image_dataset = tf.convert_to_tensor(test_image_dataset, dtype=tf.float32)
print('Testing image dataset shape: ', test_image_dataset.shape)
print(type(test_image_dataset))
print(test_image_dataset.dtype)

# Do the same for test_target
test_target = tf.convert_to_tensor(test_target, dtype=tf.uint8)
print(type(test_target))
```

```
Shape of train data:  (300, 224, 224, 3)
<class 'numpy.ndarray'>
Shape of test data:  (100, 224, 224, 3)
Shape of train target (300,)
Shape of test target (100,)
Training image dataset shape:  (300, 224, 224, 3)
```

Review Jupyter Notebook

jupyter SophiaHart_TFproj_Wildfire Last Checkpoint: 10/12/2022 (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Not Trusted | Python 3

Run | Markdown

Pretrained Model for Transfer Learning

Using Xception model from ImageNet

In [8]:

```
# Use Xception as the base model

base_model = keras.applications.xception.Xception(weights = 'imagenet', include_top=False)

base_model.summary()
```

```
 alization)                     1536)

 block14_sepconv1_act (Activati  (None, None, None,  0           ['block14_sepconv1_bn[0][0]']
 on)                            1536)

 block14_sepconv2 (SeparableCon  (None, None, None,  3159552     ['block14_sepconv1_act[0][0]']
 v2D)                           2048)

 block14_sepconv2_bn (BatchNorm  (None, None, None,  8192        ['block14_sepconv2[0][0]']
 alization)                     2048)

 block14_sepconv2_act (Activati  (None, None, None,  0           ['block14_sepconv2_bn[0][0]']
 on)                            2048)

==================================================================================================
Total params: 20,861,480
Trainable params: 20,806,952
Non-trainable params: 54,528
__________________________________________________________________________________________________
```

In [9]:

```
# Define model

n_classes = 2    # fire and nonfire

# Output from the base_model
avg = keras.layers.GlobalAveragePooling2D()(base_model.output)

# Only two classes for softmax
output = keras.layers.Dense(n_classes, activation="softmax")(avg)

# Combine base_model and softmax for our model
model = keras.Model(inputs = base_model.input, outputs = output)

# Trainable variables
print("Trainable variables ", len(model.trainable_variables))

model.summary()
```

```
Trainable variables  156
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_1 (InputLayer)           [(None, None, None,  0           []
```

Review Jupyter Notebook

jupyter SophiaHart_TFproj_Wildfire Last Checkpoint: 10/12/2022 (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Not Trusted | Python 3

Run Code

```
os.makedirs(logdir, exist_ok=True)
```

In [11]:
```
# Remove previous event files
!rm -rf ./logs/train/event*
```

In [12]:
```
tensorboard_callback = keras.callbacks.TensorBoard(logdir)
```

In [13]:
```
# Freeze the weights of the base model, ie not trainable

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

# Compile model.
# Have to use sparse_categorical_crossentropy for compile to work
optimizer = keras.optimizers.SGD(lr=0.01, momentum=0.9, decay=0.01)

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

# Train model
history = model.fit(
    train_image_dataset,
    train_target,
    epochs = 5,
    callbacks = [tensorboard_callback])
```

```
/home/shart/schoolwork/TensorFlow_Virtual/tensorflow-dev/lib/python3.7/site-packages/keras/optimizer_v2/gradient_des
cent.py:102: UserWarning: The `lr` argument is deprecated, use `learning_rate` instead.
  super(SGD, self).__init__(name, **kwargs)
```

```
Epoch 1/5
10/10 [==============================] - 25s 2s/step - loss: 0.5278 - accuracy: 0.7300
Epoch 2/5
10/10 [==============================] - 23s 2s/step - loss: 0.1841 - accuracy: 0.9233
Epoch 3/5
10/10 [==============================] - 21s 2s/step - loss: 0.1100 - accuracy: 0.9600
Epoch 4/5
10/10 [==============================] - 22s 2s/step - loss: 0.0827 - accuracy: 0.9733
Epoch 5/5
10/10 [==============================] - 22s 2s/step - loss: 0.0661 - accuracy: 0.9900
```

TensorBoard

In [14]:
```
!ls -l ./logs/train
```

```
total 404
-rw-rw-r-- 1 shart shart 413510 May 12 13:39 events.out.tfevents.1652387830.ix.11548.0.v2
```

In [15]:
```
# Show accuracy and loss with each epoch in TensorBoard

%tensorboard --logdir ./logs/train
```

Review Jupyter Notebook

jupyter SophiaHart_TFproj_Wildfire Last Checkpoint: 10/12/2022 (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Not Trusted | Python 3

Run Code

Evaluate the Model Prediction

In [16]:

```
# Evaluate the model

# Predictions is a matrix. Row is the instance, column is the output
predictions = model.predict(x=test_image_dataset, batch_size=60)
print(predictions)

# Use argmax to pull out the column with highest probability,
# which is the most probable class
predictions = tf.argmax(predictions, axis=-1)
print("Target: ", test_target)
print("Predict ", predictions)
```

```
2022-05-12 13:39:04.188907: W tensorflow/core/framework/cpu_allocator_impl.cc:82] Allocation of 364984320 exceeds 10% of free system memory.
2022-05-12 13:39:04.286464: W tensorflow/core/framework/cpu_allocator_impl.cc:82] Allocation of 364984320 exceeds 10% of free system memory.
2022-05-12 13:39:04.420577: W tensorflow/core/framework/cpu_allocator_impl.cc:82] Allocation of 364984320 exceeds 10% of free system memory.
2022-05-12 13:39:08.776628: W tensorflow/core/framework/cpu_allocator_impl.cc:82] Allocation of 243322880 exceeds 10% of free system memory.
2022-05-12 13:39:08.872625: W tensorflow/core/framework/cpu_allocator_impl.cc:82] Allocation of 243322880 exceeds 10% of free system memory.
```

```
[[1.12584266e-05 9.99988794e-01]
 [3.56126511e-05 9.99964356e-01]
 [6.39429723e-04 9.99360621e-01]
 [3.52095551e-04 9.99647856e-01]
 [9.92571414e-01 7.42854131e-03]
 [9.86689687e-01 1.33102601e-02]
 [9.98064339e-01 1.93567201e-03]
 [9.98652279e-01 1.34769070e-03]
```

In [17]:

```
# Compare predictions and test_target tensors to calculate accuracy

m = tf.keras.metrics.Accuracy()
m.update_state(predictions, test_target)
accuracy = m.result().numpy()

print("Prediction accurcay is: ", accuracy)

# Display confusion matrix
print("Confusion Matrix below: ")

tf.math.confusion_matrix(test_target, predictions)
```

```
Prediction accurcay is:  0.99
Confusion Matrix below:
```

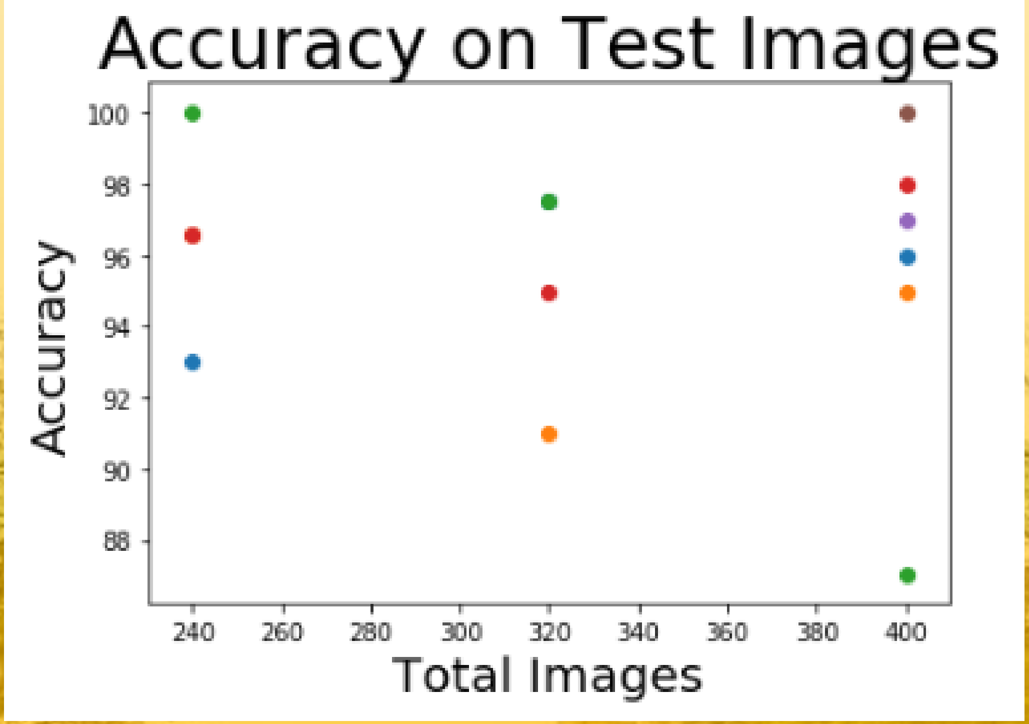
Out[17]:

```
<tf.Tensor: shape=(2, 2), dtype=int32, numpy=
array([[51,  1],
       [ 0, 48]], dtype=int32)>
```

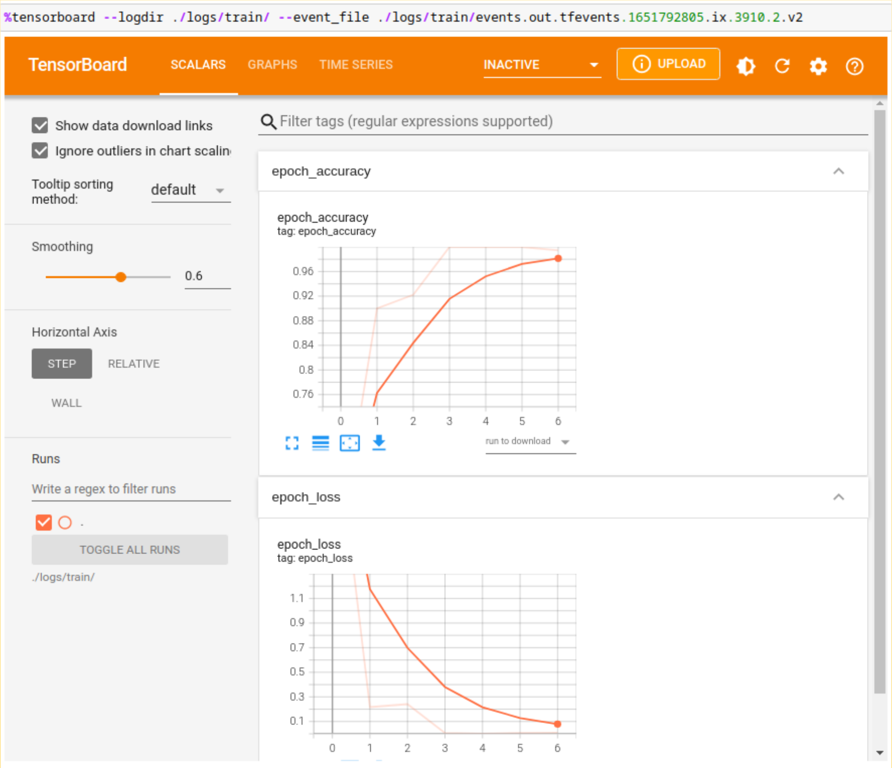
Precision = 1
Recall = 0.98

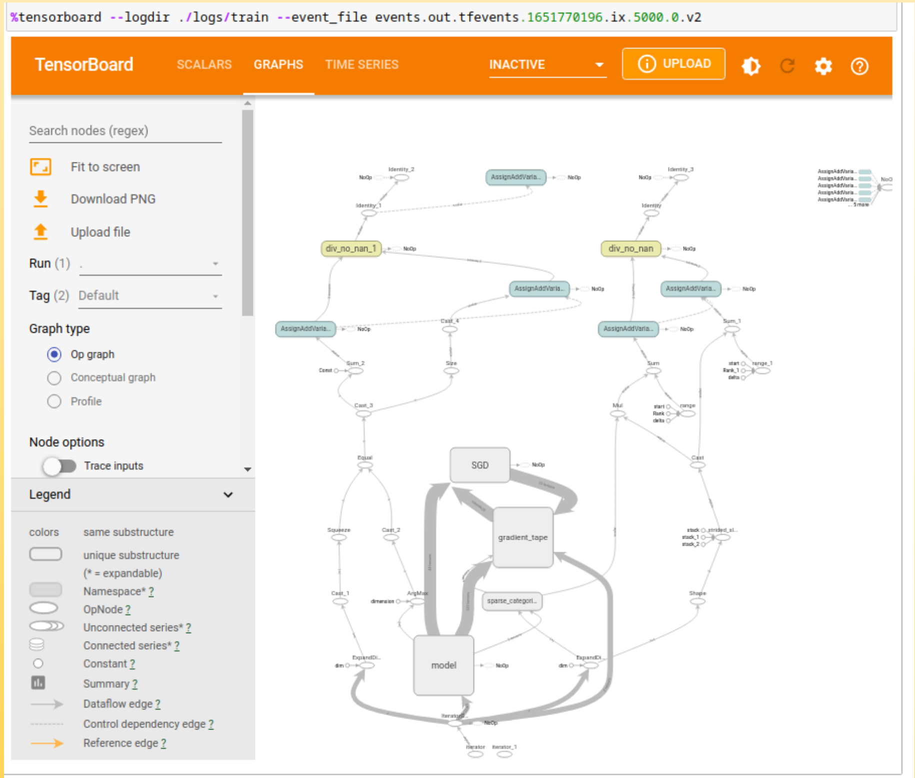
Column = prediction
Row = target
Starts from 0

Model achieves 95.7% test accuracy over 12 runs



Accuracy improves over time in learning





Model Fine-tuning

1. Unfreeze top 22 layers, trainable weights

2. Add a dropout layer on the top layer
   (don't need dropout if Batch Normalization)

3. Change optimizer (SGD → Adam)

→ **Didn't improve accuracy**

Conclusion

Computer vision with transfer learning is helpful in speed, accuracy and simplicity. And it requires less training data.

The more data we have, the wider distribution the data can be. So the two categories don't have to have a clear cut (i.e. fire and no-fire images look drastically different) to have outstanding prediction accuracy.

Machine Learning can be implemented to detect wildfire to flag the Information for human review and fire management decisions. This saves time and money, and ensures early detection which is important to control the spread of wildfire.

Future Work

Add more data.
No data augmentation.
Implement with higher computational power (e.g. Colab)

References

Aurélien Géron, *Hands-on Machine Learning with Scikit-Learn, Keras & TensorFlow, 2 Ed.* O'Reilly

Abrahams, Hafner, Erwitt, Scarpinelli, TensorFlow for Machine Intelligence, Blending Edge Press

Franc¸ois Chollet, *Xception: Deep Learning with Depthwise Separable Convolutions*, CVPR, Computer Vision Foundation Open Access version

https://theaisummer.com/receptive-field/

https://www.analyticsvidhya.com/blog/2022/01/convolutional-neural-network-an-overview/

https://towardsdatascience.com/4-pre-trained-cnn-models-to-use-for-computer-vision-with-transfer-learning-885cb1b2dfc

https://towardsdatascience.com/batch-normalization-the-greatest-breakthrough-in-deep-learning-77e64909d81d

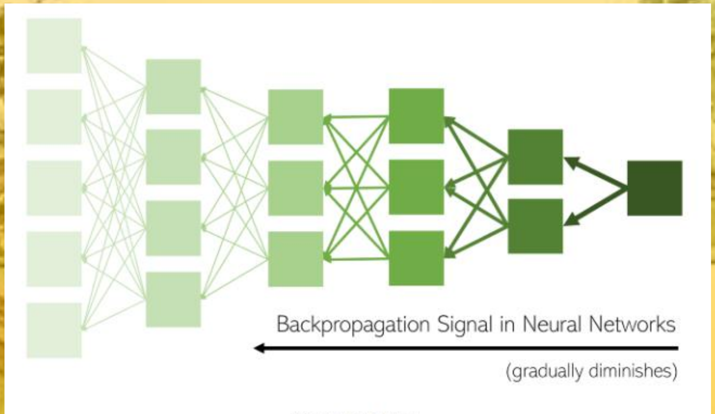
Q&A

Thank you.

Appendix A

Batch Normalization for Deep Learning

Batch Normalization solves gradient vanishing and exploding problems. One of the greatest breakthrough in deep learning development.



Created by Author

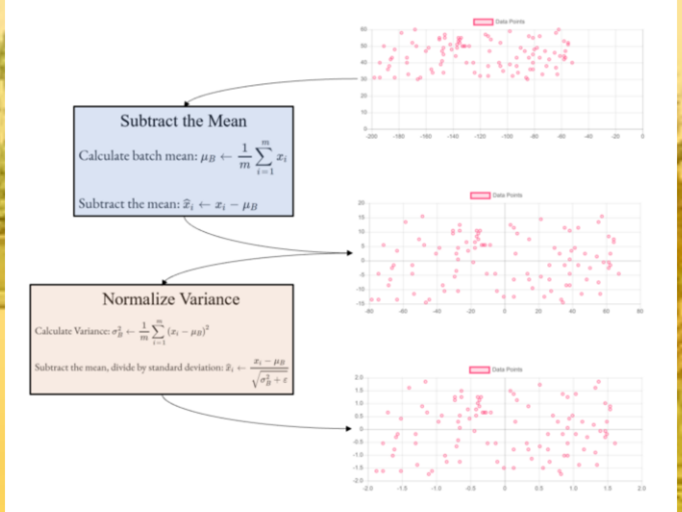
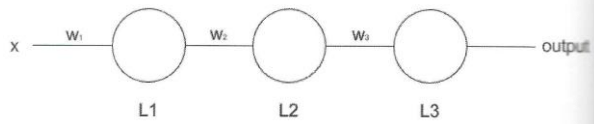


Image created by author.

Reference

Appendix B:
Chain Rule for Back Propagation

Let's assume a really simply network, with one input, one output, and two hidden layers with a single neuron. Both hidden and output neurons will be sigmoids and the loss will be calculated using cross entropy. Such a network should look like:



Let's define $L1$ as the output of first hidden layer, $L2$ the output of the second, and $L3$ the final output of the network:

$$L1 = sigmoid(w_1 \, . \, x)$$

$$L2 = sigmoid(w_2 \, . \, L1)$$

$$L3 = sigmoid(w_3 \, . \, L2)$$

Finally, the loss of the network will be:

$$loss = cross\_entropy(L3, y_{expected})$$

To run one step of gradient decent, we need to calcuate the partial derivatives of the loss function with respect of the three weights in the network. We will start from the output layer weights, applying the chain rule:

$$\frac{\partial loss}{\partial w_3} = cross\_entropy'(L3, y_{expected}) \, . \, sigmoid'(w_3 \, . \, L2) \, . \, L2$$

$L2$ is just a constant for this case as it doesn't depend on w_3

To simplify the expression we could define:

$$loss' = cross\_entropy'(L3, y_{expected})$$

$$L3' = sigmoid'(w_3 \, . \, L2)$$

The resulting expression for the partial derivative would be:

$$\frac{\partial loss}{\partial w_3} = loss' \, . \, L3' \, . \, L2$$

Now let's calculate the derivative for the second hidden layer weight, w_2:

$$L2' = sigmoid'(w_2 \, . \, L1)$$

$$\frac{\partial loss}{\partial w_2} = loss' \, . \, L3' \, . \, L2' \, . \, L1$$

And finally the derivative for w_1:

$$L1' = sigmoid'(w_1 \, . \, x)$$

$$\frac{\partial loss}{\partial w_1} = loss' \, . \, L3' \, . \, L2' \, . \, L1' \, . \, x$$

You should notice a pattern. The derivative on each layer is the product of the derivatives of the layers after it by the output of the layer before. That's the magic of the chain rule and what the algorithm takes advantage of.

We go forward from the inputs calculating the outputs of each hidden layer up to the output layer. Then we start calculating derivatives going backwards through the hidden

Abrahams, Hafner, Erwitt, Scarpinelli, *TensorFlow for Machine Intelligence*