A comparison of Tensorflow and PyTorch from the perspective of a beginner

Kevin M. Loew

July 23, 2019

Outline

- Background
  - What are TensorFlow and pyTorch?
- Analysis
  - Why?
  - What data did I use?
  - Structure of network
  - How I ran and compared
  - Transfer learning approach
- Results
  - Accuracy
  - Training Time
  - Other considerations
    - Ease of use
    - Documentation
    - job listings

Who am I?

Email: kevin.m.loew@gmail.com
Linkedin: www.linkedin.com/in/kmloew
GitHub: https://github.com/kmloew

Bachelor of Science

Embry-Riddle Aeronautical University

Research: Gravitational-Waves (LIGO)

Master of Science

Brandeis University

Research: High-Energy (CERN)

Doctor of Philosophy (Fall 2019)

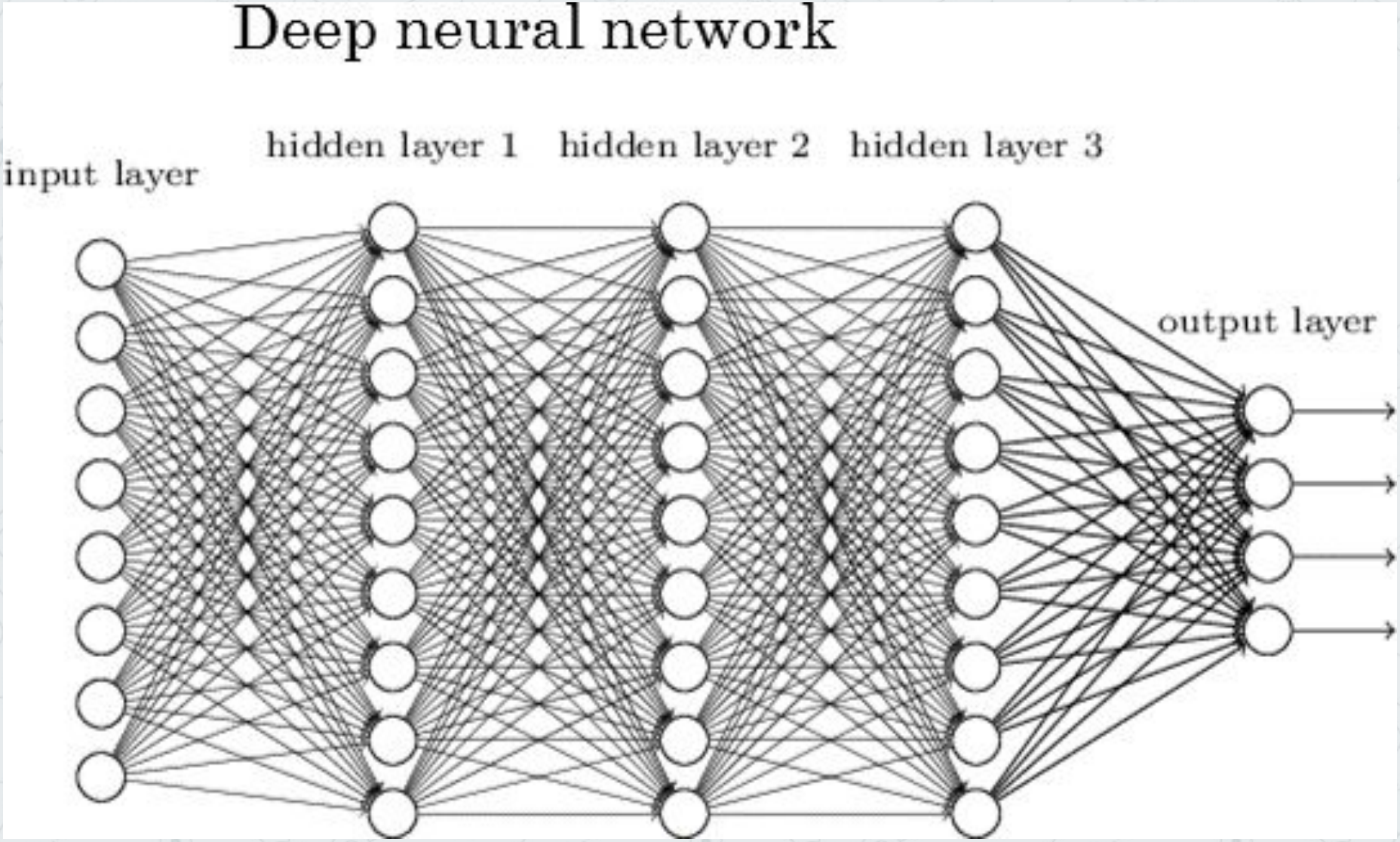
Colorado State University

Research: Pattern Formation and nanostructures (Bradley Theory Group)

This project can be found at: GitHub: https://github.com/KMLoew/DL_framework_comparison/tree/master

The data is from https://www.kaggle.com/puneet6060/intel-image-classification

Artificial Neural Network



https://datawarrior.files.wordpress.com/2016/04/slide5.png?w=542

Artificial Neural Network

- Learn tasks with few (if any) specific rules
  - Like image classification
- Collection of artificial neurons and connections
- Multiple frameworks
  - TensorFlow
  - PyTorch
  - Sonnet
  - Gluon
  - Swift
  - etc...

What are TensorFlow and PyTorch

TensorFlow vs. PyTorch

- Built on Theano
  - Google
- Static Graph
  - Keras is now part of TensorFlow and bypasses most of this
- More documentation!
- Lots of tutorials

- Extended from Torch
  - Facebook
- Dynamic graph
- More “pythonic”
- Different tools
- Becoming more popular on Udemy and online courses

Which one?

- No clear answer...
- Different tools...

https://previews.123rf.com/images/chromaco/chromaco1208/chromaco120800019/14842297-cartoon-image-of-home-repair-tools-hammer-wrench-and-screwdriver.jpg

Bake Off!

- Solve the same problem in both
- Follow tutorials and examples to build and train simple network
- After 15 epochs what is the accuracy?
- Transfer learning?
  - Resnet50 pretrained on ImageNet data
  - 10 epochs→unfreeze→20 epochs

Intel Image Classification

https://www.kaggle.com/puneet6060/intel-image-classification

Building

Forest

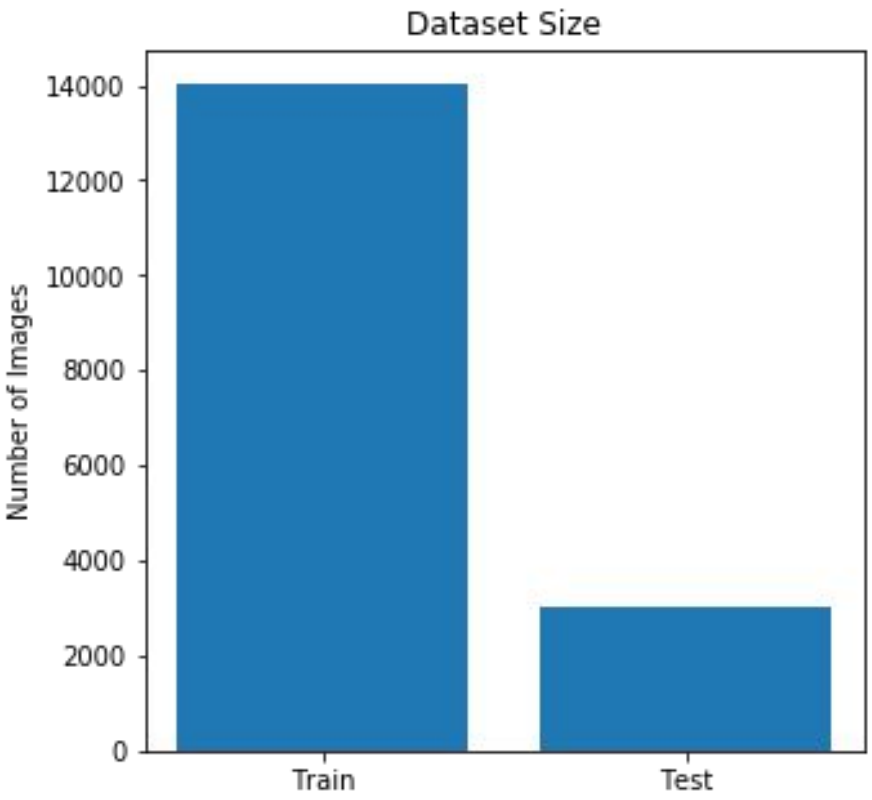
Glacier

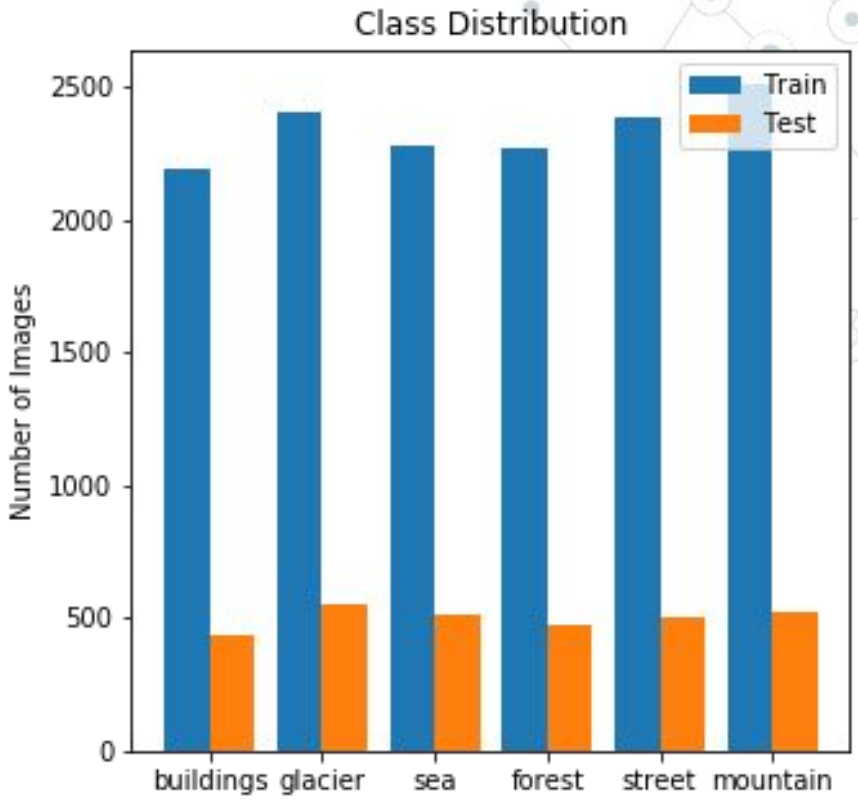
Mountain

Sea

Street

Intel Image Classification





- ◎ 6 Classes
- ◎ Fairly evenly distributed
- ◎ Not all images are the same size

Hardware

- HP Omen
  - Ubuntu 18.04
  - Python: 3.6.8
  - TensorFlow: 1.13.1
  - Torch: 1.1.0
    - Run locally
    - No other applications running

Processor: Intel i7 8750H

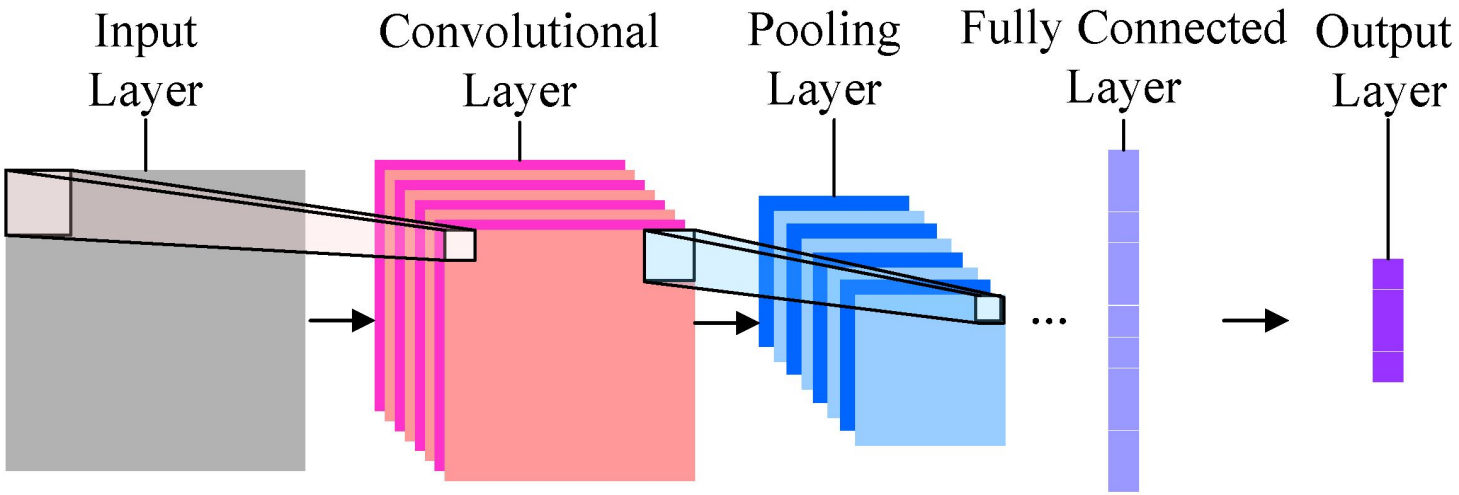
Graphics: Nvidia GeForce GTX 1070

RAM: 32GB of RAM

Storage: 512GB SSD + 1TB Hard Drive

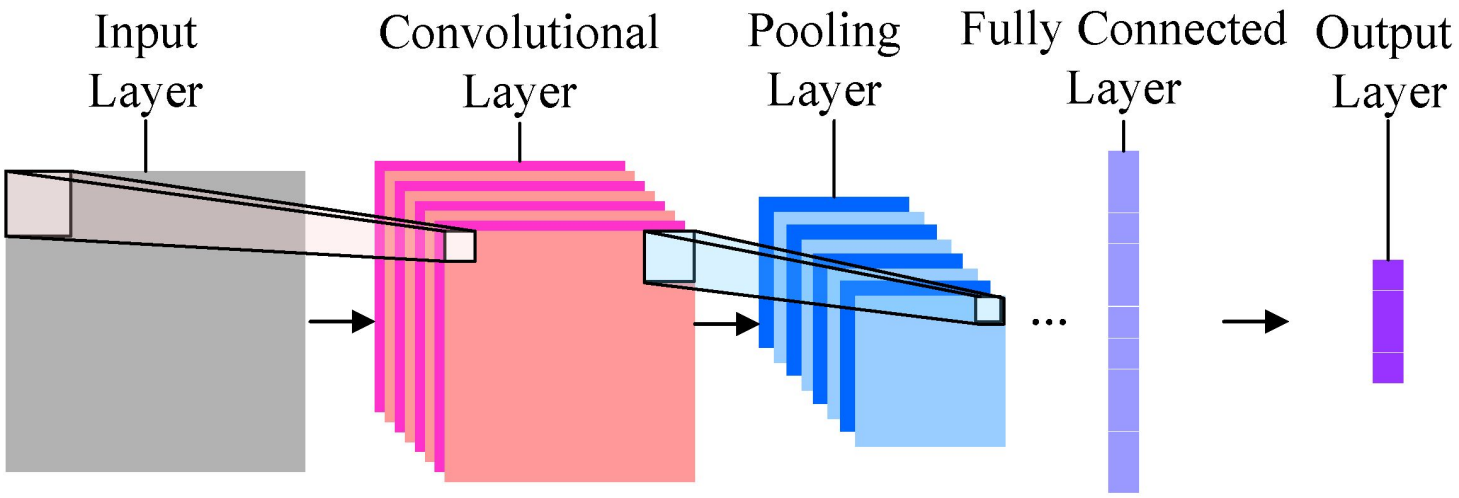
Simple Network

- Convolutional Layer
- Pooling Layer
- Convolutional Layer
- Pooling Layer
- Flattening Layer
- Dense (Fully connected) Layer
- Dense (Fully connected) Layer
- Dropout: 50% dropped connections
- Dense (Fully connected) Layer



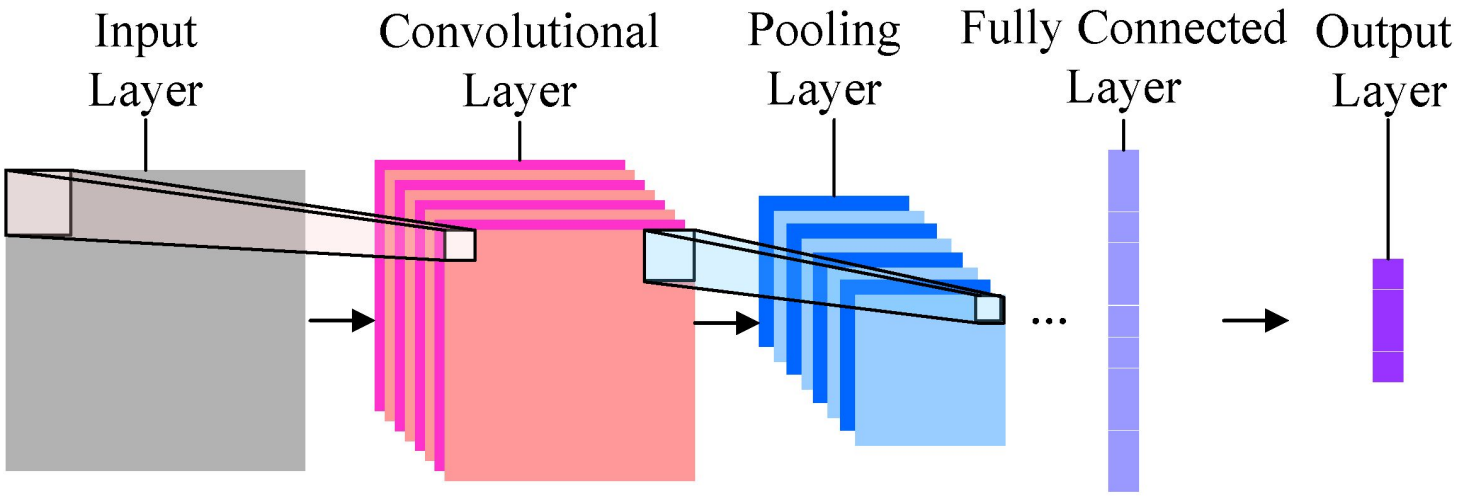
Training

- Accuracy as metric
- Sparse Categorical Cross Entropy as loss function
- Adam optimizer with learning rate of 0.001
  - fastai has variable learning rate
  - For the fine tuning of the TF resnet, the learning rate was 0.0001
- Batch size of 64



Data Augmentation

- Resize images to be 150x150
- Normalize pixel range to [0,1]
- Randomly rotate up to 25°
- Randomly flip images horizontally and vertically



TensorFlow+Keras Layers

```
model=keras.Sequential()

model.add(Conv2D(32, kernel_size=(3,3), strides=2, activation='relu', input_shape=(150,150,3)))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Conv2D(64, kernel_size=(3,3), strides=2, activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Flatten())
model.add(Dense(128,activation='relu'))
model.add(Dense(64,activation='relu'))
model.add(Dropout(rate=0.5))
model.add(Dense(len(classes), activation='softmax'))
```

- ◎ Very easy to build
- ◎ Lots of examples and documentation

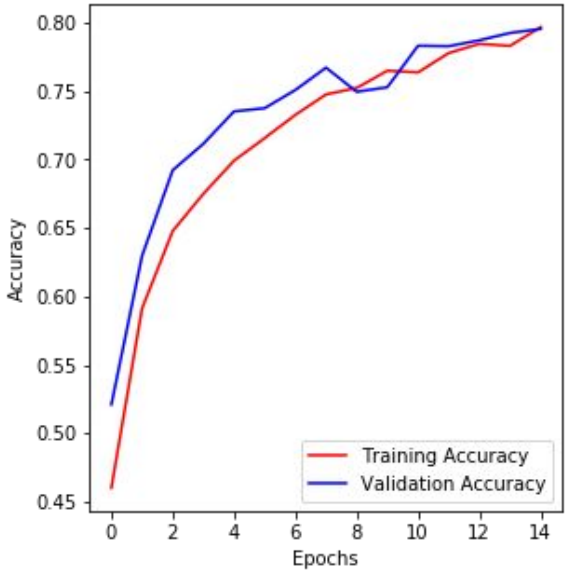
PyTorch+nn Layers

```python
class myModel(nn.Module):
    def __init__(self):
        super(myModel,self).__init__()

        self.cnn1=nn.Conv2d(in_channels=3, out_channels=32, kernel_size=3,stride=2)
        self.relu=nn.ReLU()
        self.maxpool1=nn.MaxPool2d(kernel_size=2)

        self.cnn2=nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3, stride=2)
        self.maxpool2=nn.MaxPool2d(kernel_size=2)

        self.fc1=nn.Linear(in_features=64*9*9, out_features=128)
        self.fc2=nn.Linear(in_features=128, out_features=64)
        self.droput=nn.Dropout(p=0.5)
        self.fc3=nn.Linear(in_features=64, out_features=len(classes))

    def forward(self,x):
        out=self.cnn1(x)
        out=self.relu(out)
        out=self.maxpool1(out)
        out=self.cnn2(out)
        out=self.relu(out)
        out=self.maxpool2(out)
        out=out.view(64,64*9*9)
        out=self.fc1(out)
        out=self.relu(out)
        out=self.fc2(out)
        out=self.relu(out)
        out=self.droput(out)
        out=self.fc3(out)
        return out
```

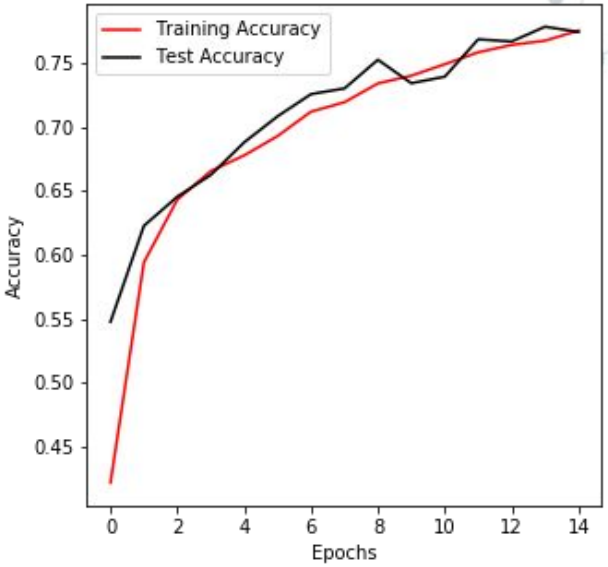
◎ Need to define more, harder to figure out...

Results: Training plots

TensorFlow

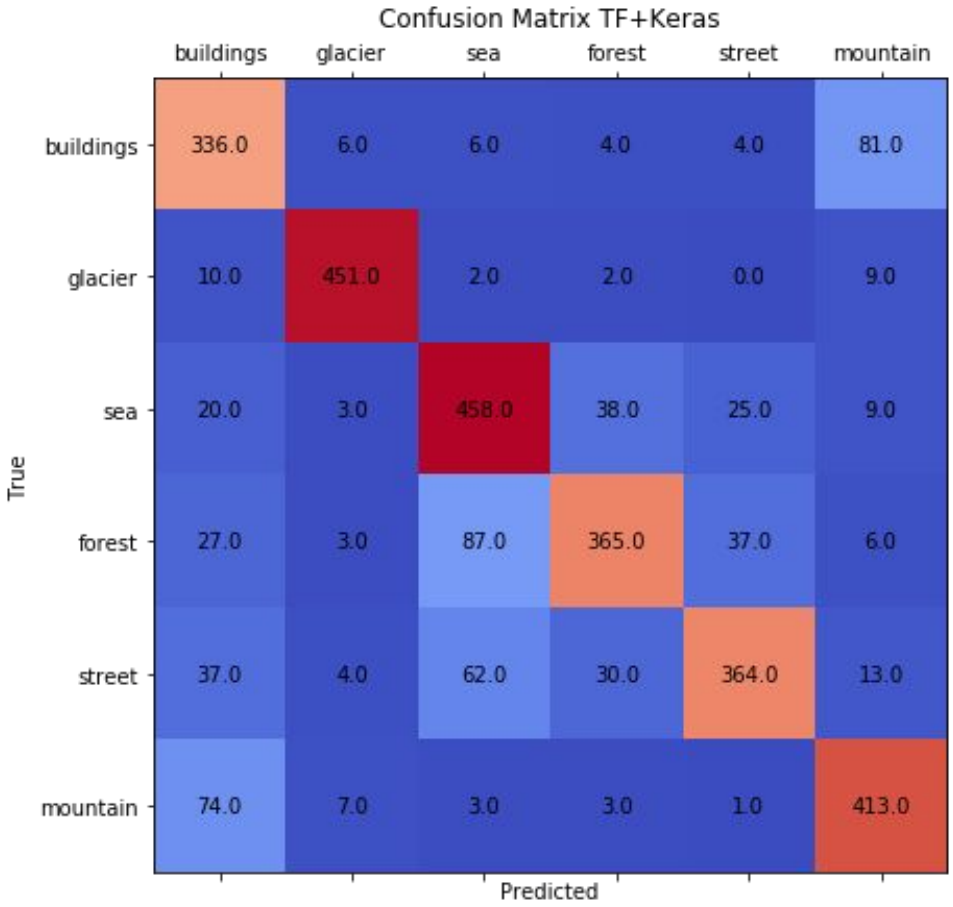


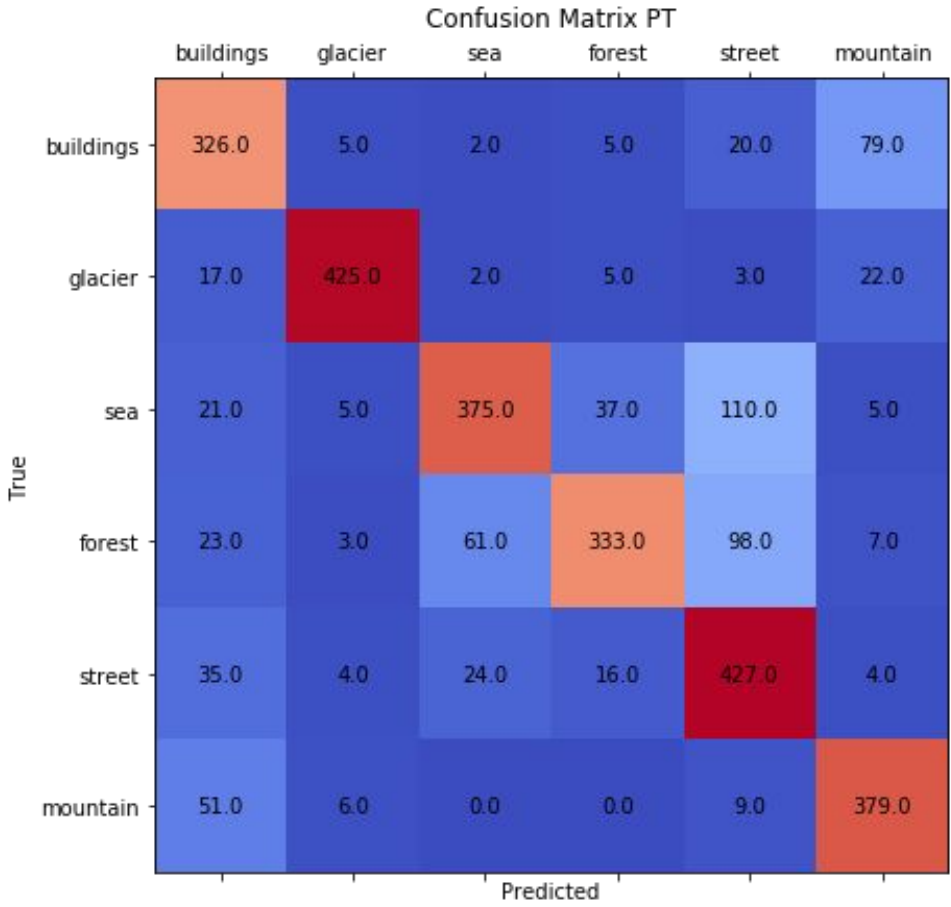
PyTorch



- ◎ Similar accuracy curves
- ◎ TensorFlow+Keras achieve higher accuracy

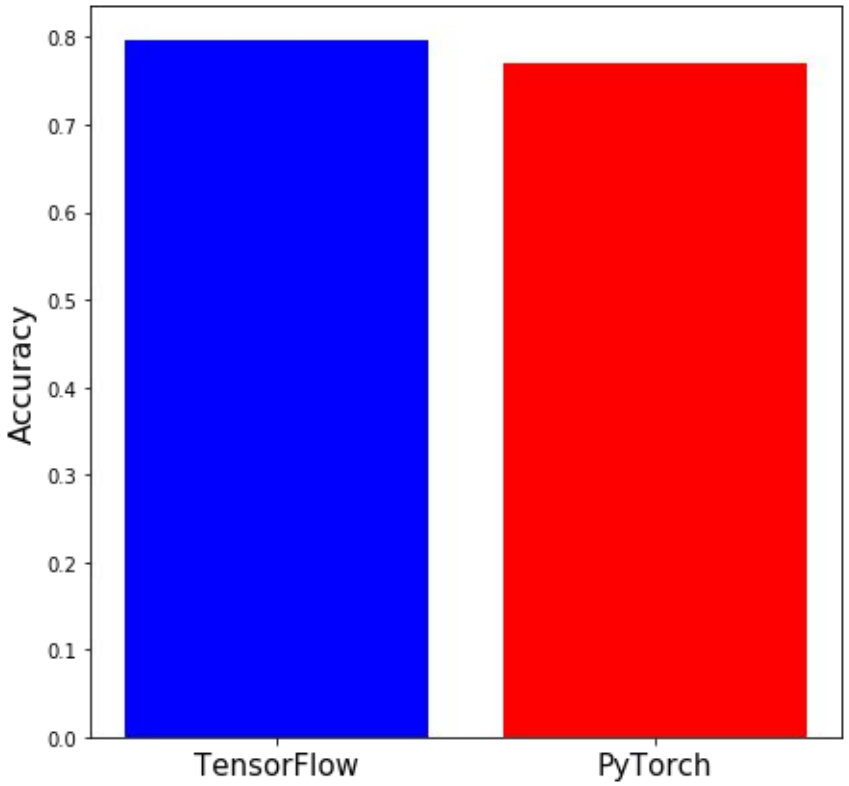
Results: Confusion Matrix

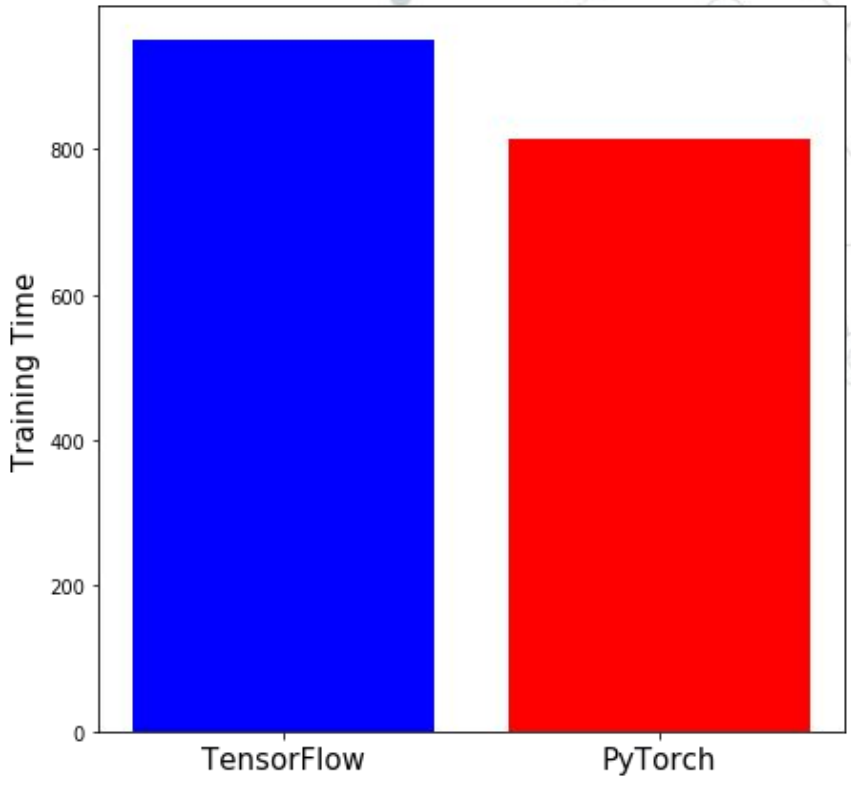




◎ Tensorflow with Keras layers does a bit better

Results: Accuracy and Training Time





- TensorFlow was a bit more accurate
  - 80% vs 77%
- PyTorch trained slightly faster
- Keras Layers were easier to work with

Transfer Learning

- Often instead of building a new network, an old one will be retrained
- This speeds up training because it already knows how to extract information, just needs to learn what you want
- “Cut off” top layers and place new ones

- I chose resnet50 as my base model
- Pretrained on Imagenet data (~14 million images)

TensorFlow

```
base_model=ResNet50(input_shape=(150,150,3),include_top=False)
```

```
model=keras.Sequential([
  base_model,
  GlobalAveragePooling2D(),
  Dense(len(classes),activation='softmax')
])

model.summary()
```

- Need to declare what layers are trainable
  - base_model.trainable=False
- Not too bad

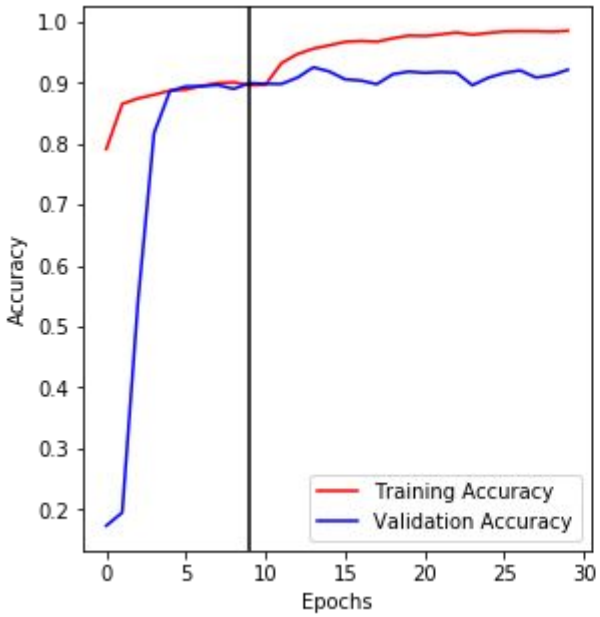
Pytorch+fastai

```
1 model=cnn_learner(data,models.resnet50,metrics=accuracy,model_dir="/tmp/model/")
```

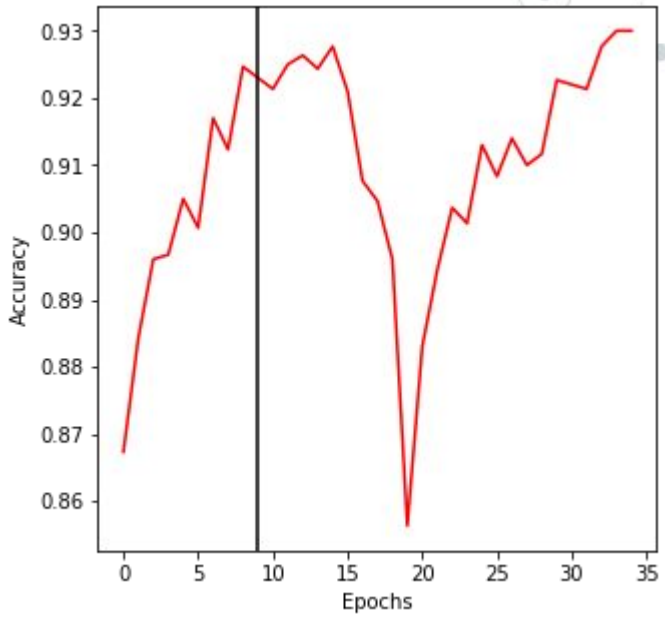
◎ So easy!

Results: Training plots

TensorFlow

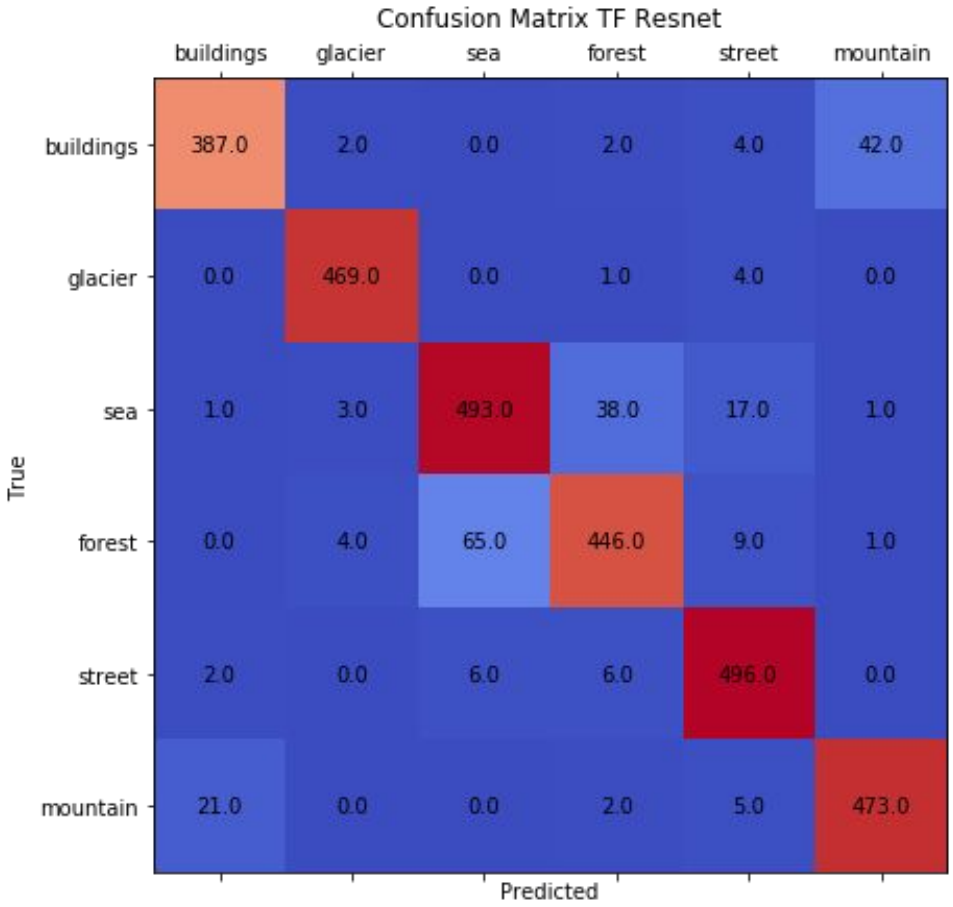


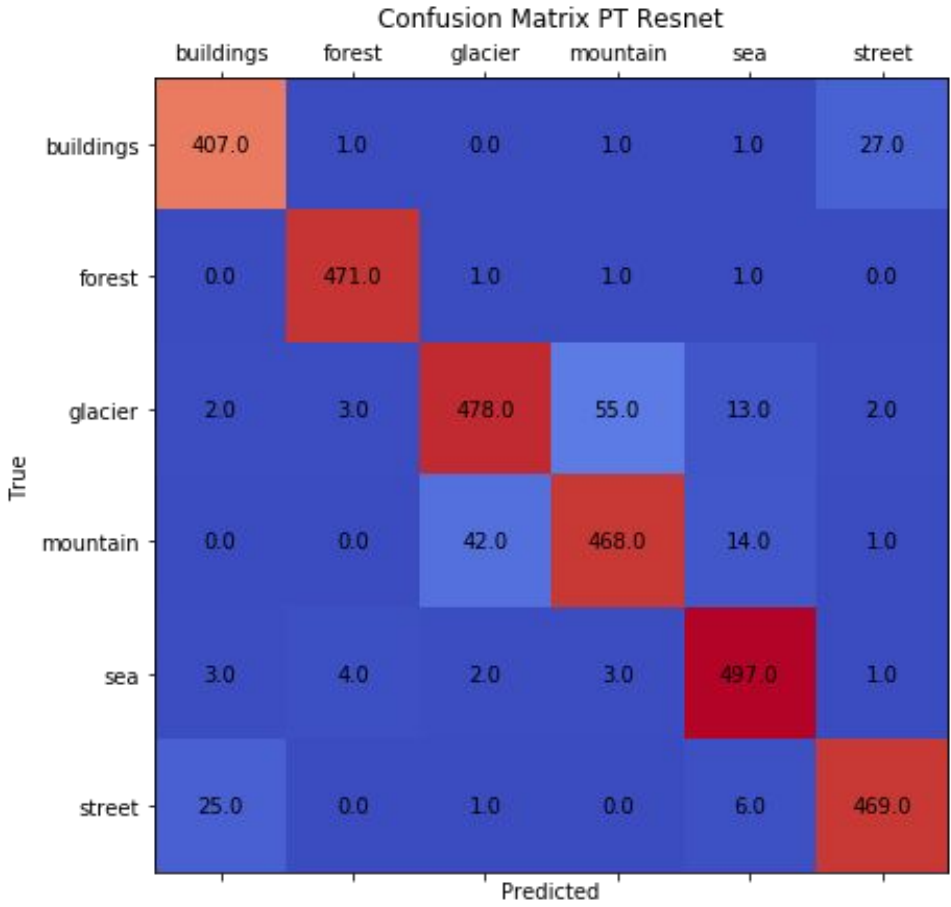
PyTorch



- Both get higher accuracy than the simple model
- PyTorch+fastai has more fluctuation for fine tuning

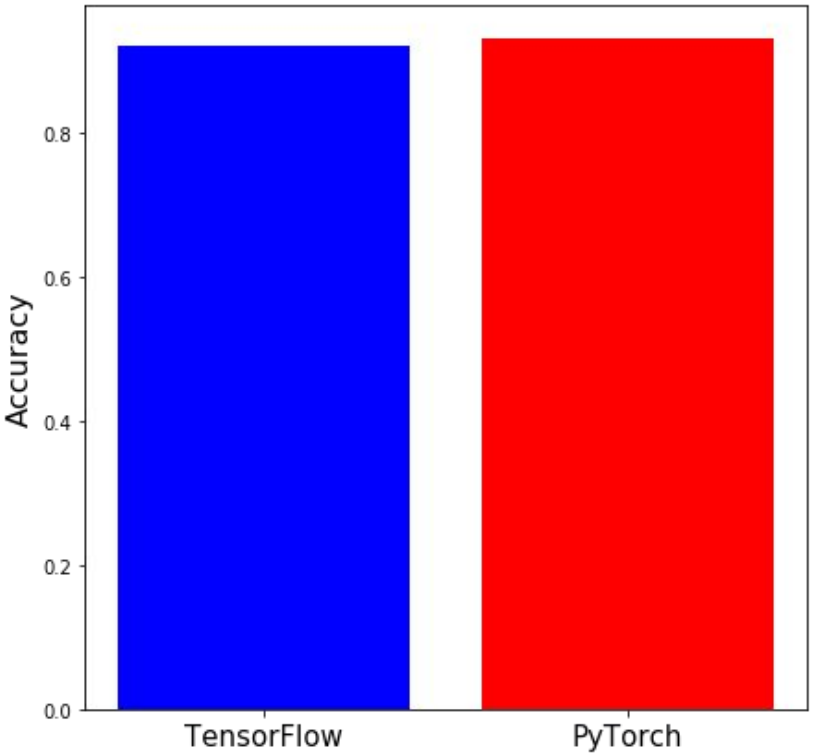
Results: Confusion Matrix

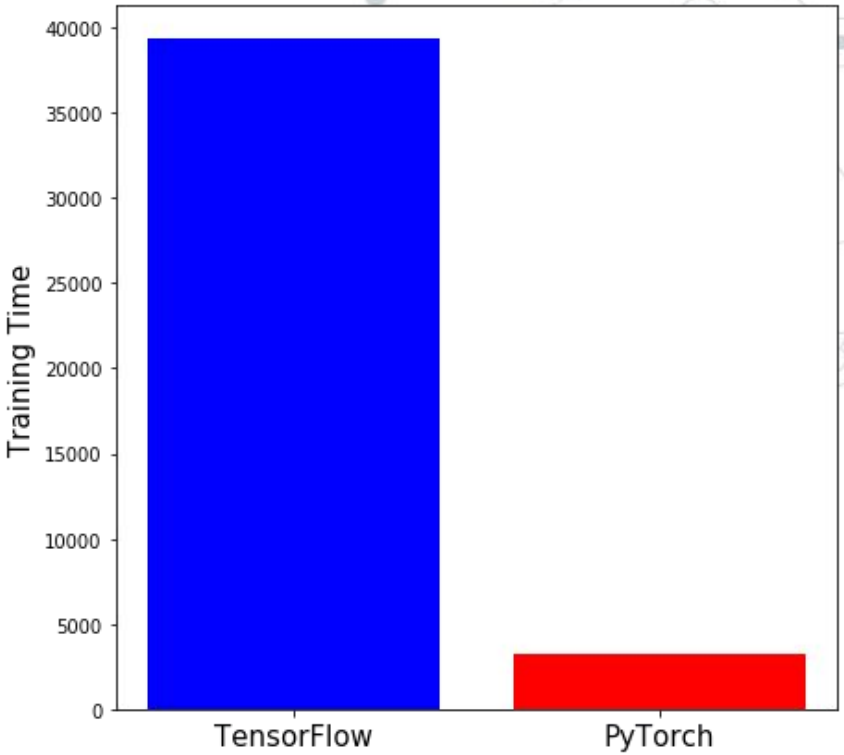




◎ Very similar

Results: Accuracy and Training Time





- Virtually Identical accuracy
  - 92% vs 93%
- The difference is in training time...

Final Results

Simple

- TensorFlow+Keras Layers were easier to build
- TensorFlow+Keras got higher final accuracy
- PyTorch Trained Faster
- Data handling was easier in PyTorch

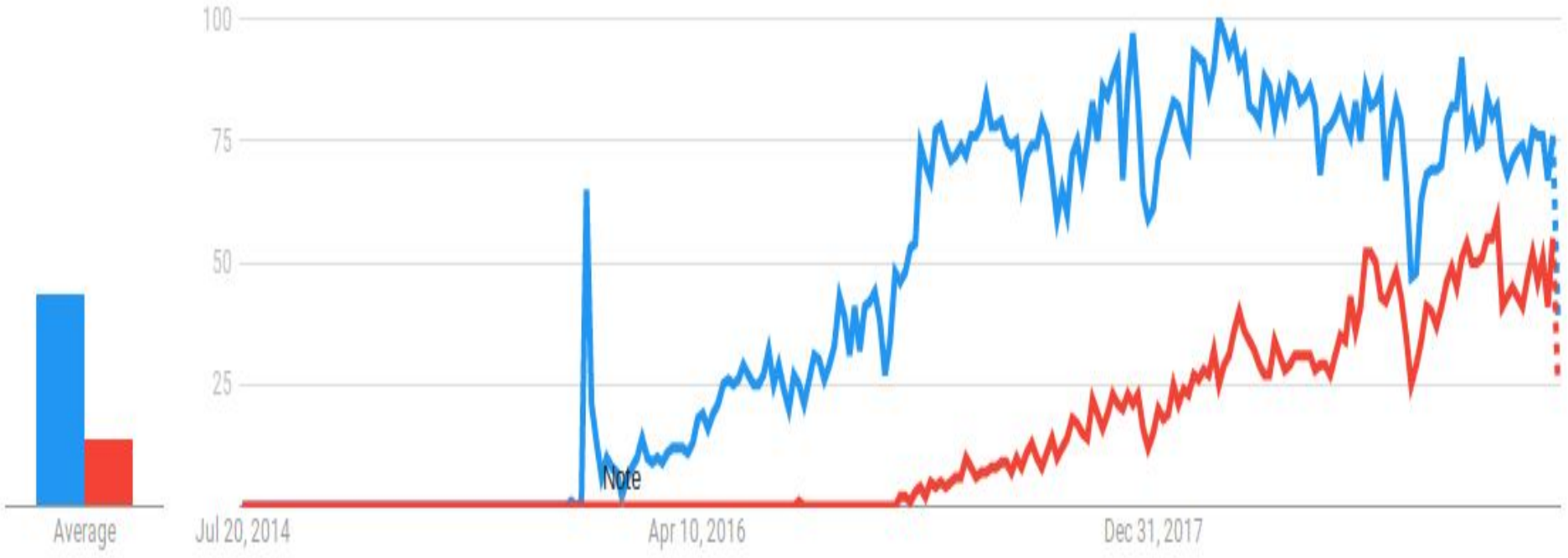
PreTrained

- PyTorch+fastai was significantly faster
- PyTorch+fastai was easier to work with, once I figured it out

Other thoughts?

- Which one is used in industry?
- Amount of documentation?
- Which is easier to learn?

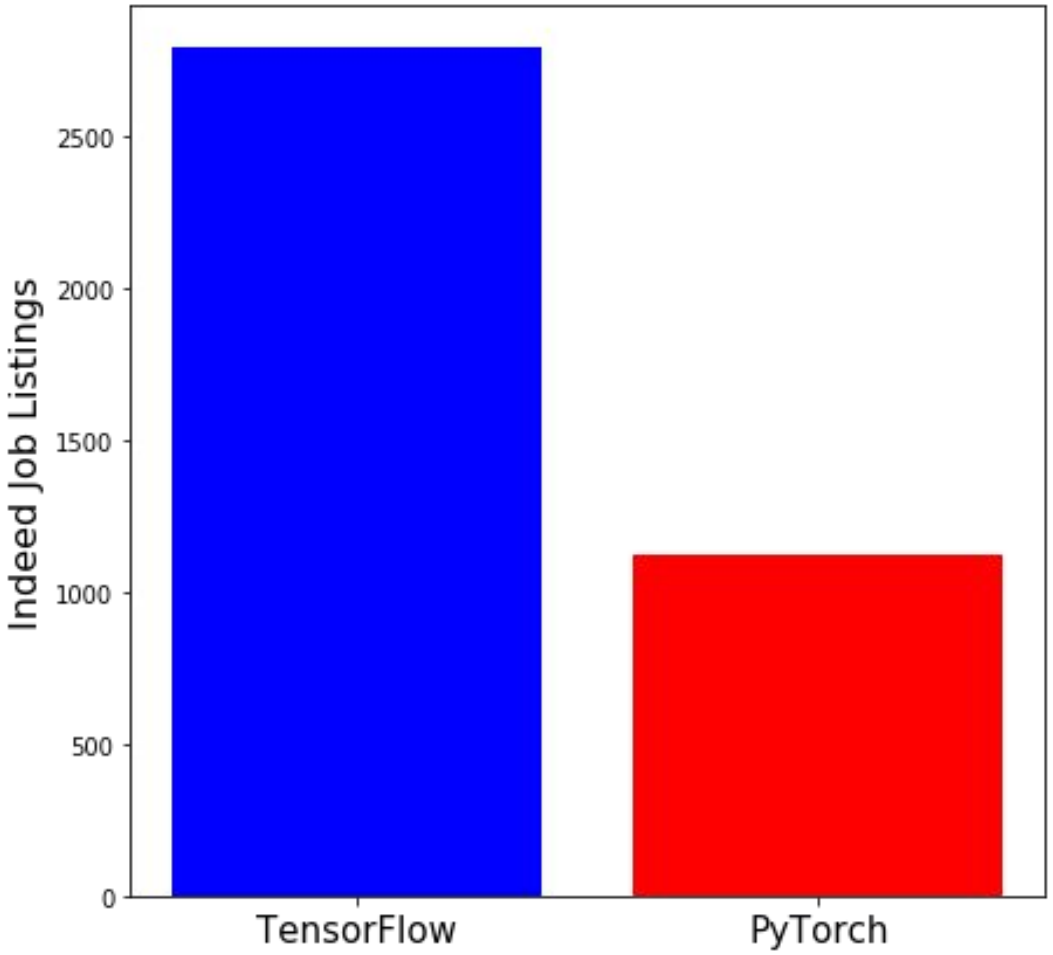
Google Trends



TensorFlow

PyTorch

Indeed Job Postings



◎ Same trend as google

Conclusion

- Use the tool that makes sense for you
- TensorFlow is more popular and has more documentation
  - I would recommend this since it seems to be more common
- PyTorch is easy to pick up and experiment with
  - Good for learning concepts
- Fastai does amazing for transfer learning!