NAS-FPN: Learning Scalable Feature Pyramid Architecture for Object Detection

Golnaz Ghaisi, Tsung-Yi Lin, Ruoming Pang, Quoc V. Le
Published April 16th 2019

Presented by Andrew Pang
Objective

● Convolutional architectures for object detection are typically designed manually.
● The main objective is to learn a better architecture of feature pyramid networks for object detection.
● The resulting architecture is named NAS-FPN, short for Neural Architecture Search - Feature Pyramid Network.
Introduction

● Feature Pyramid Network (FPN):
  ○ Represents an image with multiscale feature layers
  ○ Consists of a backbone model which helps structure the feature pyramid
  ○ Adjacent layers in the feature hierarchy of the backbone model are combined with top-down and lateral connections.
  ○ High level features are upsampled and combined with high resolution features

● Neural Architecture Search (NAS):
  ○ Reinforcement Learning using RNN controller
  ○ Controller designs cells or layers to obtain a network
Methodology

- Based on RetinaNet framework, consisting of two main components:
  - Backbone Network
  - Feature Pyramid Network
- FPN takes multiscale feature layers as inputs and generates output feature layers in identical scales.
- The inputs and outputs are of the same scale, allowing for the FPN to be stacked repeatedly.
Methodology

- The pyramid network consists of a series of merging cells that introduce cross-scale connections.
- Each cell has 4 prediction steps made by the RNN controller:
  a. Select first feature layer
  b. Select second feature layer w/o replacement
  c. Select output feature resolution
  d. Select binary op to combine selected feature layers and generate output
- Output layer is pushed back into the stack
Experiments

- Separated into parts:
  - Finding the right NAS to create a RNN controller that will discover the NAS-FPN architecture
  - Test the discovered NAS-FPN with different backbone models and image sizes
  - Run analysis of accuracy and speed

- Architecture Search:
  - Proxy Task used to speed up training
  - RNN controller uses average precision (AP) on validation set as reward.

- Implementation:
  - Use RetinaNet for experiments
  - 50 epoch training on TPUs with 64-image batches from COCO dataset.
Experiments

- Discovered FPN architectures
  - Starts with vanilla FPN
  - As AP increases, the controller is able to figure out useful cross-scale connections
  - Better feature reuse is implemented as the controller converges.
Results and Analysis

- Tests were run with different components being adjusted:
  - Stacking different number of pyramid networks
  - Adopting different backbone architectures
  - Adjusting feature dimension of FPNs

![Graphs showing AP vs. FLOPs for different components:](image)

(a) Number of pyramid networks  
(b) Backbone architectures  
(c) Feature dimension
Results and Analysis

- Which model is best depends on the user’s needs and priority.
- Accurate models created and tested against FPN baseline.
Results and Analysis

- NAS-FPNLite was designed for more lightweight tasks at higher speeds.
- Tested against FPNLite and SSDLite
## Results and Analysis

<table>
<thead>
<tr>
<th>model</th>
<th>image size</th>
<th># FLOPs</th>
<th># params</th>
<th>inference time (ms)</th>
<th>test-dev AP</th>
</tr>
</thead>
<tbody>
<tr>
<td>YOLOv3 DarkNet-53 [40]</td>
<td>320 x 320</td>
<td>38.97 B</td>
<td>-</td>
<td>22 (Titan X)</td>
<td>28.2</td>
</tr>
<tr>
<td>MobileNetV2 + SSDLite [36]</td>
<td>320 x 320</td>
<td>1.61 B</td>
<td>4.3M</td>
<td>200 (Pixel 1 CPU)</td>
<td>22.1</td>
</tr>
<tr>
<td>MnasNet + SSDLite [36]</td>
<td>320 x 320</td>
<td>1.4 B</td>
<td>4.3M</td>
<td>190 (Pixel 1 CPU)</td>
<td>22.3</td>
</tr>
<tr>
<td>MnasNet-92 + SSDLite [36]</td>
<td>320 x 320</td>
<td>2.08 B</td>
<td>5.3M</td>
<td>227 (Pixel 1 CPU)</td>
<td>22.9</td>
</tr>
<tr>
<td>FPNLite MobileNetV2 @ 64</td>
<td>320 x 320</td>
<td>1.51 B</td>
<td>2.02M</td>
<td>192 (Pixel 1 CPU)</td>
<td>22.7</td>
</tr>
<tr>
<td>FPNLite MobileNetV2 @ 128</td>
<td>320 x 320</td>
<td>2.03 B</td>
<td>2.20M</td>
<td>264 (Pixel 1 CPU)</td>
<td>24.3</td>
</tr>
<tr>
<td>NAS-FPNLite MobileNetV2 (3 @ 48)</td>
<td>320 x 320</td>
<td>1.52 B</td>
<td>2.16 M</td>
<td>210 (Pixel 1 CPU)</td>
<td>24.2</td>
</tr>
<tr>
<td>NAS-FPNLite MobileNetV2 (7 @ 64)</td>
<td>320 x 320</td>
<td>1.96 B</td>
<td>2.62 M</td>
<td>285 (Pixel 1 CPU)</td>
<td>25.7</td>
</tr>
<tr>
<td>YOLOv3 DarkNet-53 [40]</td>
<td>608 x 608</td>
<td>140.69 B</td>
<td>-</td>
<td>51 (Titan X)</td>
<td>33.0</td>
</tr>
<tr>
<td>CornerNet Hourglass [18]</td>
<td>512 x 512</td>
<td>-</td>
<td>-</td>
<td>244 (Titan X)</td>
<td>40.5</td>
</tr>
<tr>
<td>Mask R-CNN X-152-32x8d [11]</td>
<td>1280 x 800</td>
<td>-</td>
<td>-</td>
<td>125 (P100)</td>
<td>45.2</td>
</tr>
<tr>
<td>RetinaNet R-101 [81]</td>
<td>832 x 500</td>
<td>-</td>
<td>-</td>
<td>90 (Titan X)</td>
<td>34.4</td>
</tr>
<tr>
<td>FPN R-50 @256 [23]</td>
<td>640 x 640</td>
<td>193.68 B</td>
<td>34.0M</td>
<td>37.5 (P100)</td>
<td>37.0</td>
</tr>
<tr>
<td>FPN R-101 @256 [23]</td>
<td>640 x 640</td>
<td>254.2B</td>
<td>53.0M</td>
<td>51.1 (P100)</td>
<td>37.8</td>
</tr>
<tr>
<td>FPN R-50 @256 [23]</td>
<td>1024 x 1024</td>
<td>495.8B</td>
<td>34.0M</td>
<td>73.0 (P100)</td>
<td>40.1</td>
</tr>
<tr>
<td>FPN R-101 @256 [23]</td>
<td>1024 x 1024</td>
<td>651.1B</td>
<td>53.0M</td>
<td>83.7 (P100)</td>
<td>41.1</td>
</tr>
<tr>
<td>FPN AmeobaNet @256 [23]</td>
<td>1280 x 1280</td>
<td>1311 B</td>
<td>114.4 M</td>
<td>210.4 (P100)</td>
<td>43.4</td>
</tr>
<tr>
<td>NAS-FPN R-50 (7 @ 256)</td>
<td>640 x 640</td>
<td>281.3B</td>
<td>60.3M</td>
<td>56.1 (P100)</td>
<td>39.9</td>
</tr>
<tr>
<td>NAS-FPN R-50 (7 @ 256)</td>
<td>1024 x 1024</td>
<td>720.4B</td>
<td>60.3M</td>
<td>92.1 (P100)</td>
<td>44.2</td>
</tr>
<tr>
<td>NAS-FPN R-50 (7 @ 256)</td>
<td>1280 x 1280</td>
<td>1125.5B</td>
<td>60.3M</td>
<td>131.9 (P100)</td>
<td>44.8</td>
</tr>
<tr>
<td>NAS-FPN R-50 (7 @ 384)</td>
<td>1280 x 1280</td>
<td>2086.3B</td>
<td>103.9M</td>
<td>192.3 (P100)</td>
<td>45.4</td>
</tr>
<tr>
<td>NAS-FPN R-50 (7 @ 384) + DropBlock</td>
<td>1280 x 1280</td>
<td>2086.3B</td>
<td>103.9M</td>
<td>192.3 (P100)</td>
<td>46.6</td>
</tr>
<tr>
<td>NAS-FPN AmeobaNet (7 @ 384)</td>
<td>1280 x 1280</td>
<td>2633 B</td>
<td>166.5 M</td>
<td>278.9 (P100)</td>
<td>48.0</td>
</tr>
<tr>
<td>NAS-FPN AmeobaNet (7 @ 384) + DropBlock</td>
<td>1280 x 1280</td>
<td>2633 B</td>
<td>166.5 M</td>
<td>278.9 (P100)</td>
<td>48.3</td>
</tr>
</tbody>
</table>

Table 1: Performance of RetinaNet with NAS-FPN and other state-of-the-art detectors on test-dev set of COCO.

NAS-FPN: Learning Scalable Feature Pyramid Architecture for Object Detection
Advantages and Disadvantages

- **Advantages:**
  - Stacked pyramid networks allows for anytime detection
  - Developed NAS-FPN architecture tends to have higher AP compared to other models with similar workload.

- **Disadvantage:**
  - Often times uses more FLOPs and Parameters
  - Tends to be a bit slower
Takeaway

- Using a neural architecture search to design a feature pyramid network can lead to better results than a traditional FPN.
- Overall, the NAS-FPN architecture seems to perform better than the other detectors tested against it.
- Offers improvement on any applications that require or use object detection.
Thanks for listening

Any Questions?