# Latency is also Important: Comparing MobileNetV2 and a YOLO-style Architecture for Deployment to Jetson Nano

Regan Willis

I trained a two face detection models based off the MobileNetV2 architecture and YOLO architecture. The MobileNetV2 architecture is created directly from their paper's architecture. The YOLO architecture is altered to better fit an edge device, reducing the layers and dimensionality as is done in Fast YOLO.
I ran hyperparameter optimization on both architectures to find the ideal learning rate and batch size on these architectures with the provided dataset. Both models are deployed to a Jetson Nano for an accuracy and latency comparison.

**MobileNetV2 Architecture**
I created a PyTorch module based off the architecture of MobileNetV2 as explained in their paper. MobileNetV2 starts with a convolutional layer. It then uses bottleneck residual blocks, which are a group of convolutional layers with the middle layer expanding to a much larger dimensionality than the input and output based on scaling factor. These inner layers use depthwise separable convolutions, which lowers the computational cost. The architecture used batch normalization and ReLU. The resulting model is 3,538,213 parameters.

**YOLO Architecture**
I also created a model based off of early YOLO architecture. The model takes its architecture from YOLO version 1 and adds batch normalization as used in YOLO version 2 (YOLO9000). I included a learning rate scheduler that allows the weights to warm up and increases the learning rate towards the end of training, as recommended in the YOLO paper. This model was very large at over 600,000,000 parameters. This is unrealistic for deployment to edge devices, so architecture changes were made.
The YOLOv1 paper mentions a Fast YOLO model that is only 9 convolutional layers, compared to YOLO's 24, and has smaller filter sizes. The paper did not show which layers were removed or what new filter size was used. I removed some of the repetitive convolutional layers and divided the filter size by half. The model is still much larger than MobileNetV2 at 213,763,493 parameters. See the architecture changes in below:

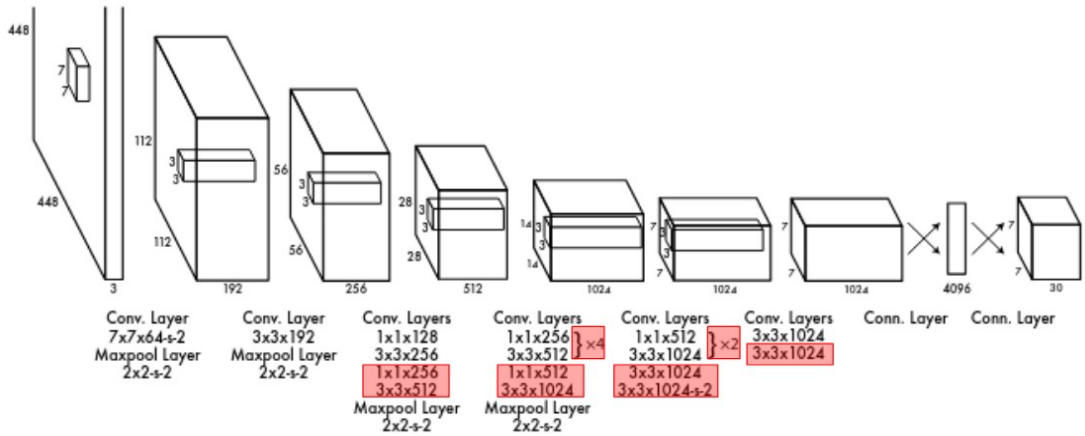


*Figure 1: YOLO architecture (original figure from https://arxiv.org/pdf/1506.02640) with removed layers shown in red boxes.*

**Hyperparameter Optimization**

I used Hyperopt to search for optimal parameters for my models. I searched over the batch size to see if a smaller or larger batch size would be better for these architectures and dataset. I also searched over learning rate.

| Parameter | Values |
|---|---|
| Batch size | 16, 32, 64 |
| Learning rate - MobileNetV2 | 1e-4 - 1e-3 |
| Learning rate - YOLO | 1e-5 - 1e-2 |

I searched 10 different evaluations for each model architecture. To save compute, I first trained the models to 20 epochs and analyzed the loss scores. The models with the highest loss scores were further trained to 100 epochs. The hyperparameter optimization table is shown in Table 1 in the Appendix.

**Measuring Performance – Mean Average Precision**
I computed the mean average precision (mAP) at intersection over union (IoU) of 50% after every validation to track model performance. Since the validation dataset may include multiple bounding boxes per image but my model will only predict one, I used the IoU of the closest bounding box in my calculation. I calculate the IoU of the predicted bounding box and ground truth bounding box with the provided calculate_iou function. If the IoU is over 50%, that prediction is added to my true positive count. Otherwise, I add that prediction to my false positive count. After iterating over the validation dataset and predictions, I compute the precision as the number of true positives over the number of true positives and false positives.

**Performance**
The original MobileNetV2 model was trained with the Adam optimizer and a learning rate of 1e-3 for 40 epochs at a batch size of 32. The best MobileNetV2 and YOLO model after hyperparameter optimization are also included in the table below:

| Model | Best Val Loss | IoU at Best Val Loss | mAP@50 at Best Val Loss | mAP@50 on Test Dataset |
|---|---|---|---|---|
| mobilenet_baseline | 15.26085376739502 | 57.2677166071184 | 74.38155106014489 | **44.247** |
| mobilenetv2_1744571778.807785 | 17.2798 | 55.23755809982462 | 75.61829894000475 | 40.503 |
| yolo_1744574759.3731036 | 16.1658 | 55.08814121733498 | 69.43455954070545 | 42.069 |

The YOLO model validation loss stopped improving at 55 epochs. MobileNetV2 kept improving until epoch 93. Both models scored lower than the baseline MobileNetV2 model, which did not use any hyperparameter optimization.

**Deploying to Jetson Nano**
To deploy to Jetson Nano, I first convert the trained PyTorch model to ONNX format. Then the model is compiled with TensorRT on the Nano. Note there was some issue with deploying the YOLO model to Jetson Nano. The model was formatted to ONNX but a TRT model could not be generated due to some parsing error. The accuracy and latency comparision is shown in the table below:

| Model | mAP@50 on Test Dataset (%) | Latency (ms) |
|---|---|---|
| mobilenet_baseline | **44.247** | 18 |
| mobilenetv2_1744571778.807785 | 40.503 | **15** |
| yolo_1744574759.3731036 | 42.069 | -- |

**Conclusion**

Both the standard MobileNetV2 architecture and the YOLO-like architecture consistently perform around 30% mAP or more on the test dataset. The two architectures have comparable performance, with the YOLO architecture performing slightly better. This could be due to the learning rate step function used for the YOLO architecture or the greater amount of parameters.

The hyperparameter optimization did not yield better results than the baseline model. This shows that both models are robust to training within the given learning rate range and batch size options. In the future, it may be helpful to tune using a dropout layer probability or weight decay value.

The hyperparameter-optimized MobileNetV2 model shows slightly lower latency than the YOLO model. Latency metrics were not obtained for the YOLO model but they would likely be slower than MobileNetV2 due to the large number of parameters.

For real-time deployment, the baseline model (mobilenet_baseline) is suitable for a slightly higher accuracy but will also have slightly higher latency, while the hyperparameter-optimized model (mobilenetv2_1744571778.807785) will have lower accuracy but also lower latency.

## Appendix

| model_id | score | arch | learning_rate | batch_size |
|---|---|---|---|---|
| 1744568734.0887354 | 19.863113403320312 | mobilenetv2 | 0.0006875409385151203 | 32 |
| 1744569188.9165504 | 24.105188740356445 | mobilenetv2 | 0.0001136638602181646 7 | 32 |
| 1744569603.6144097 | 19.735118865966797 | mobilenetv2 | 0.0008487471886125 02 | 64 |
| 1744570047.3371203 | 24.923780441284 18 | mobilenetv2 | 0.0002100369040027708 7 | 64 |
| 1744570477.4527164 | 21.63854408264 16 | mobilenetv2 | 0.0001856331941983792 4 | 16 |
| 1744570926.7047217 | 20.414514541625977 | mobilenetv2 | 0.0005389372182431753 | 32 |
| 1744571341.2417684 | 19.26774787902832 | mobilenetv2 | 0.0003257775342156766 | 16 |
| 1744571778.807785 | 18.55148096899414 | mobilenetv2 | 0.0009422826315669756 | 32 |
| 1744572180.7024539 | 22.96359634399414 | mobilenetv2 | 0.0002259388578730529 7 | 64 |
| 1744572590.7177842 | 20.34746742248535 | mobilenetv2 | 0.0001463418650805390 5 | 16 |
| 1744573022.1955445 | 17.88639259338379 | yolo | 0.002413592311985555 | 16 |
| 1744573480.8197696 | 20.494688034057617 | yolo | 1.2245937637286573e-05 | 32 |
| 1744573902.5584137 | 17.84968376159668 | yolo | 0.0013360680949037429 5 | 64 |
| 1744574294.6865199 | 17.55556297302246 | yolo | 0.002059095681765505 | 16 |
| 1744574759.3731036 | 17.179468154907227 | yolo | 0.0017580225237505772 | 16 |
| 1744575218.4424615 | 18.07927131652832 | yolo | 0.0001368515728835524 | 32 |
| 1744575625.0224514 | 19.519210815429688 | yolo | 0.007064393063370373 | 16 |
| 1744576087.0446835 | 18.856327056884766 | yolo | 0.0001040177708487414 5 | 64 |
| 1744576496.831536 | 18.014820098876953 | yolo | 3.720026885090472e-05 | 32 |
| 1744576922.4919732 | 19.076078414916992 | yolo | 2.962102598998594e-05 | 32 |

*Table 1: Hyperparameter optimization output.*