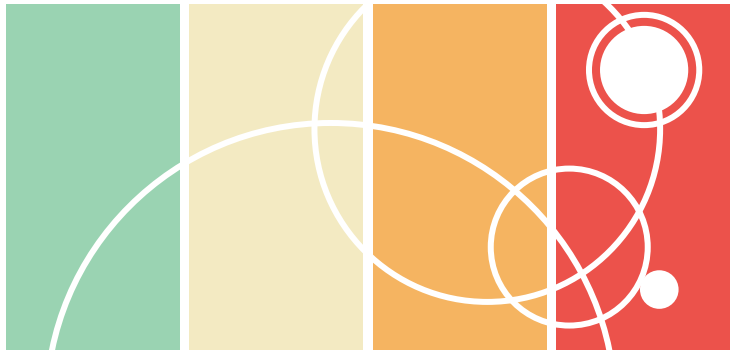


# 物体検出入門

孫 建強
どんぐり研究所

# 物体検出入門

**物体分類 – torch**

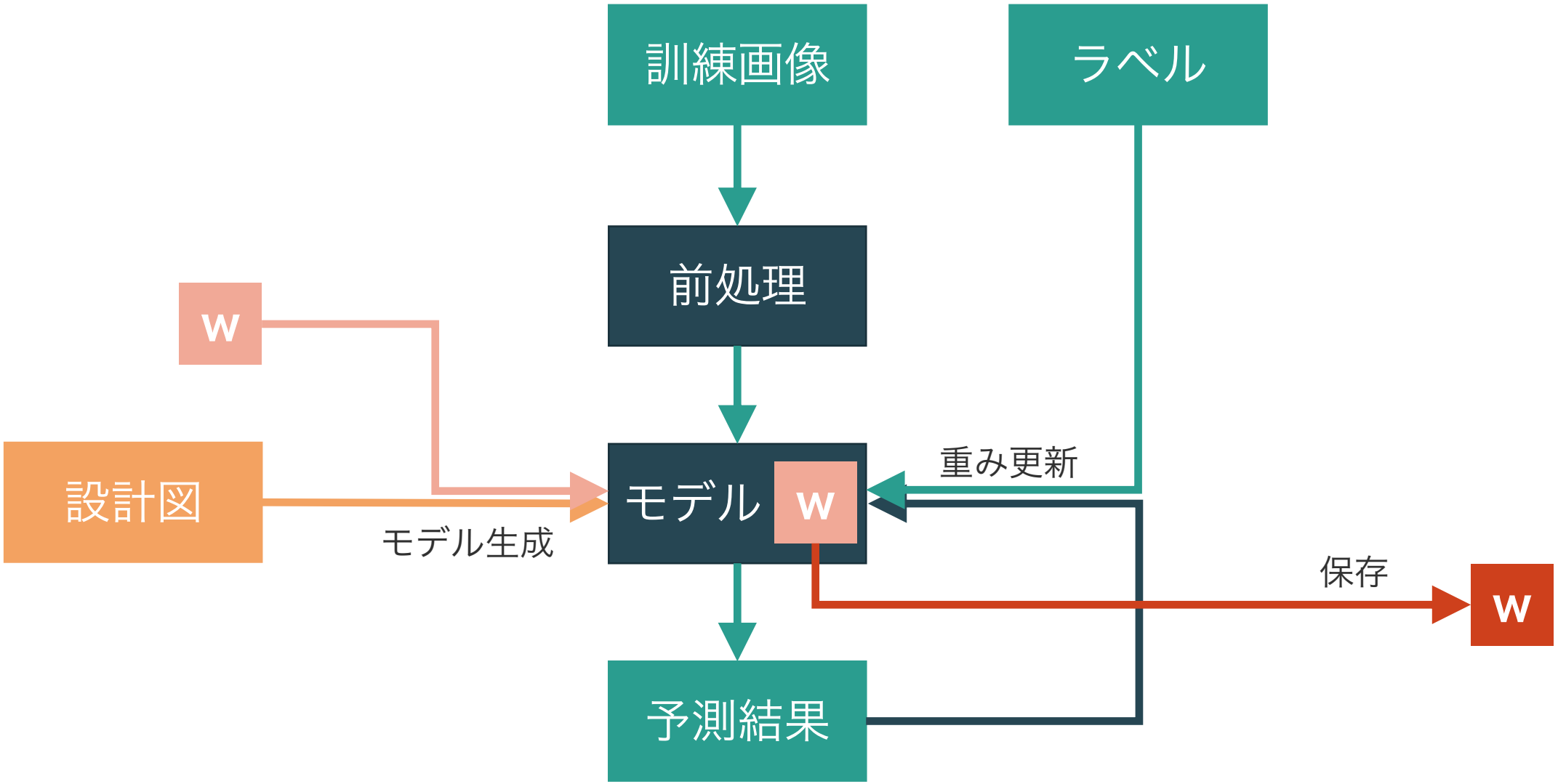
物体検出アルゴリズム

アノテーションツール

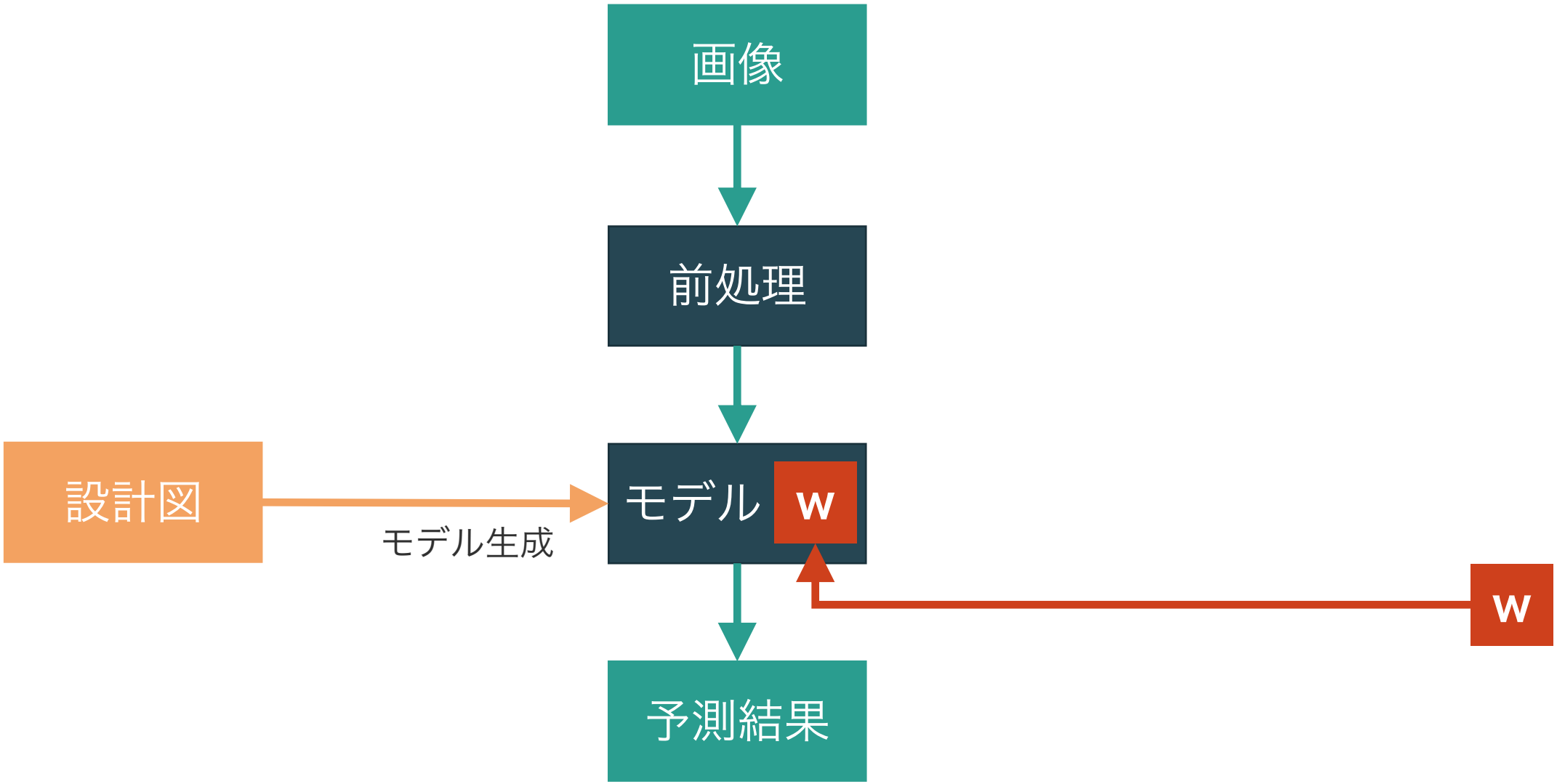
物体検出 – Detectron2

物体検出 – MMDetection

# 訓練プロセス



# 推論プロセス



# モデル設計

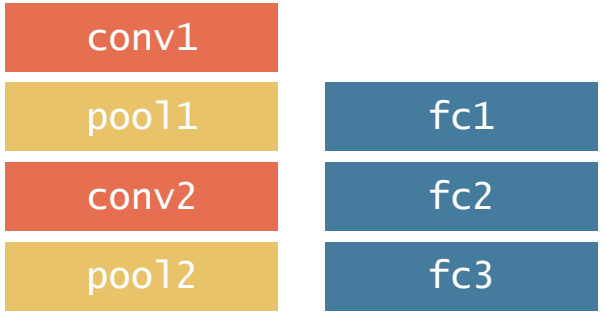
```
class SimpleNet(torch.nn.Module):


  def __init__(self):
    supper().__init__()
    self.nn_inputs = (((((224-5+1)/2)-5+1)/2)^2)*32

    self.conv1 = torch.nn.Conv2d(3, 16, 5)
    self.pool1 = torch.nn.MaxPool2d(2, 2)
    self.conv2 = torch.nn.Conv2d(16, 32, 5)
    self.pool2 = torch.nn.MaxPool2d(2, 2)
    self.fc1 = torch.nn.Linear(self.nn_inputs, 512)
    self.fc2 = torch.nn.Linear(512, 64)
    self.fc3 = torch.nn.Linear(64, 5)



  def forward(self, x):
    x = torch.nn.functional.relu(self.conv1(x))
    x = self.pool1(x)
    x = torch.nn.functional.relu(self.conv2(x))
    x = self.pool2(x)
    x = x.view(-1, self.nn_inputs)
    x = torch.nn.functional.relu(self.fc1(x))
    x = torch.nn.functional.relu(self.fc2(x))
    x = self.fc3(x)
    return x
```

# モデル設計

```
class SimpleNet(torch.nn.Module):
```

```
def __init__(self):
  supper().__init__()
  self.nn_inputs = (((((224-5+1)/2)-5+1)/2)^2)*32

  self.conv1 = torch.nn.Conv2d(3, 16, 5)
  self.pool1 = torch.nn.MaxPool2d(2, 2)
  self.conv2 = torch.nn.Conv2d(16, 32, 5)
  self.pool2 = torch.nn.MaxPool2d(2, 2)
  self.fc1 = torch.nn.Linear(self.nn_inputs, 512)
  self.fc2 = torch.nn.Linear(512, 64)
  self.fc3 = torch.nn.Linear(64, 5)
```

モデル構築に必要な部品を準備



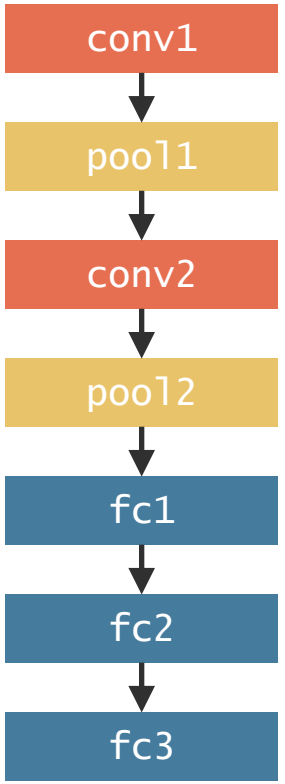

```
def forward(self, x):
  x = torch.nn.functional.relu(self.conv1(x))
  x = self.pool1(x)
  x = torch.nn.functional.relu(self.conv2(x))
  x = self.pool2(x)
  x = x.view(-1, self.nn_inputs)
  x = torch.nn.functional.relu(self.fc1(x))
  x = torch.nn.functional.relu(self.fc2(x))
  x = self.fc3(x)
  return x
```

# モデル設計

```
class SimpleNet(torch.nn.Module):


  def __init__(self):
    supper().__init__()
    self.nn_inputs = (((((224-5+1)/2)-5+1)/2)^2)*32

    self.conv1 = torch.nn.Conv2d(3, 16, 5)
    self.pool1 = torch.nn.MaxPool2d(2, 2)
    self.conv2 = torch.nn.Conv2d(16, 32, 5)
    self.pool2 = torch.nn.MaxPool2d(2, 2)
    self.fc1 = torch.nn.Linear(self.nn_inputs, 512)
    self.fc2 = torch.nn.Linear(512, 64)
    self.fc3 = torch.nn.Linear(64, 5)
```

```
def forward(self, x):
  x = torch.nn.functional.relu(self.conv1(x))
  x = self.pool1(x)
  x = torch.nn.functional.relu(self.conv2(x))
  x = self.pool2(x)
  x = x.view(-1, self.nn_inputs)
  x = torch.nn.functional.relu(self.fc1(x))
  x = torch.nn.functional.relu(self.fc2(x))
  x = self.fc3(x)
  return x
```

部品同士を繋ぎデータの流れを定義

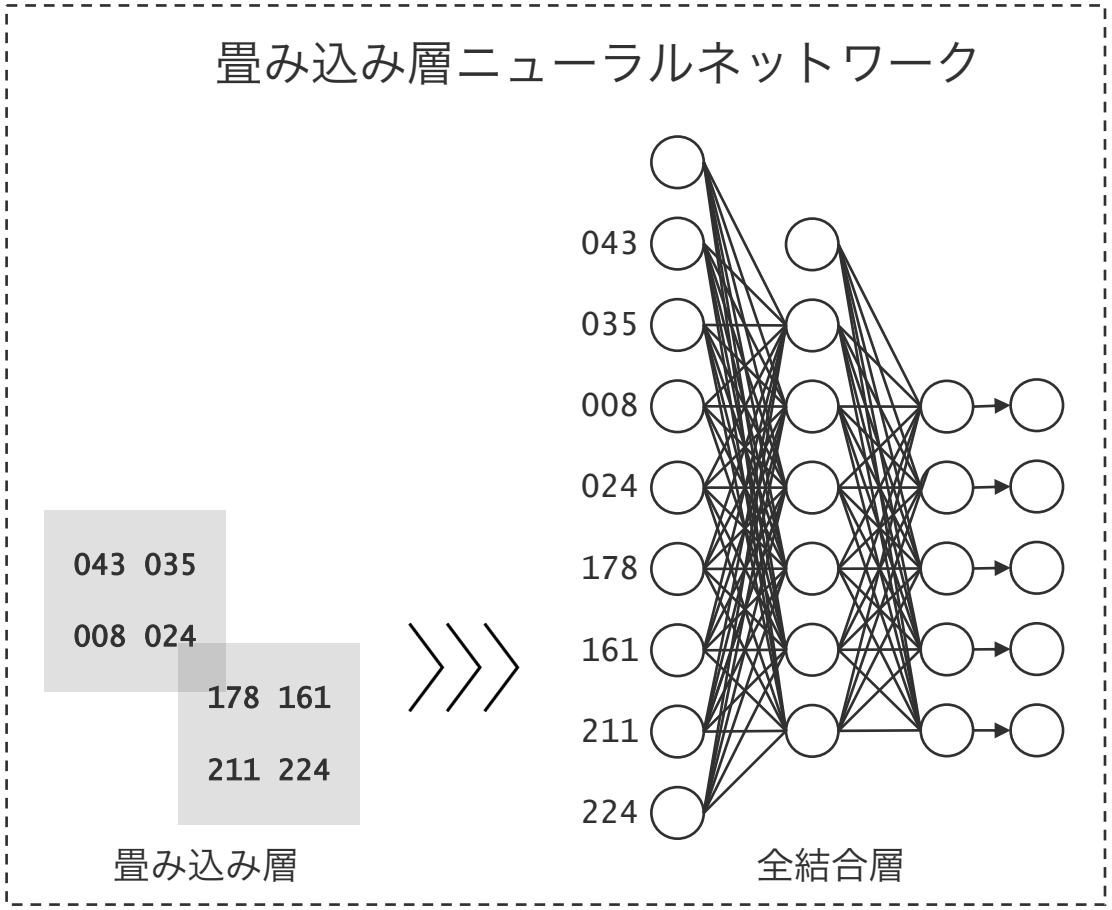


# モデル設計

```
class SimpleNet(torch.nn.Module):

  def __init__(self):
    supper().__init__()
    self.nn_inputs = (((((224-5+1)/2)-5+1)/2)^2)*32

    self.conv1 = torch.nn.Conv2d(3, 16, 5)
    self.pool1 = torch.nn.MaxPool2d(2, 2)
    self.conv2 = torch.nn.Conv2d(16, 32, 5)
    self.pool2 = torch.nn.MaxPool2d(2, 2)
    self.fc1 = torch.nn.Linear(self.nn_inputs, 512)
    self.fc2 = torch.nn.Linear(512, 64)
    self.fc3 = torch.nn.Linear(64, 5)



  def forward(self, x):
    x = torch.nn.functional.relu(self.conv1(x))
    x = self.pool1(x)
    x = torch.nn.functional.relu(self.conv2(x))
    x = self.pool2(x)
    x = x.view(-1, self.nn_inputs)
    x = torch.nn.functional.relu(self.fc1(x))
    x = torch.nn.functional.relu(self.fc2(x))
    x = self.fc3(x)
    return x
```

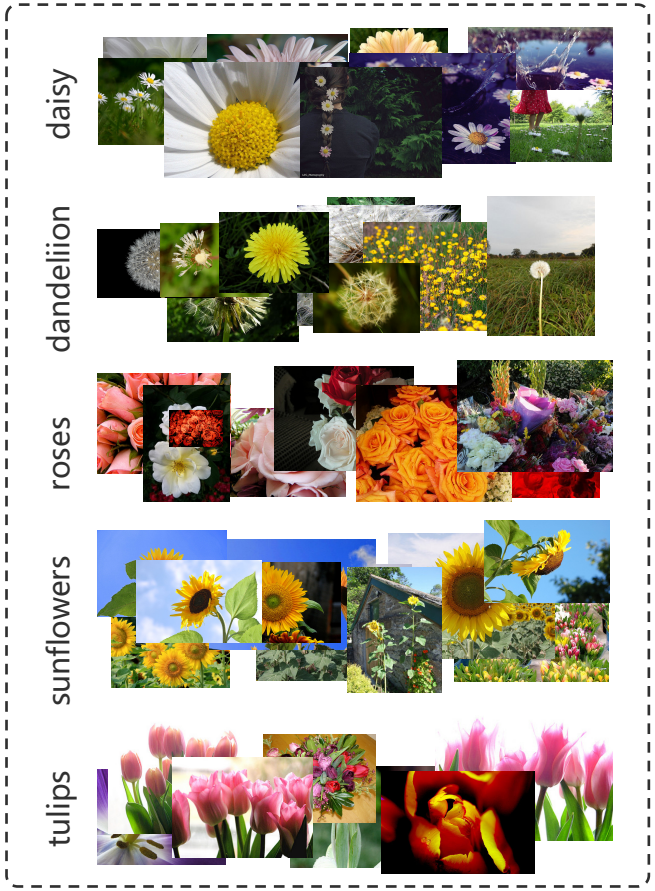
畳み込み層の出力数を計算

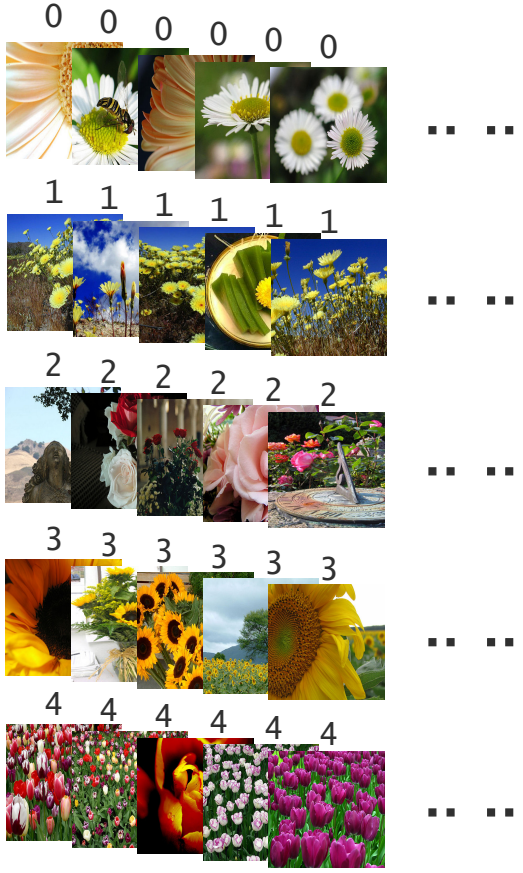
畳み込み層が出力する
行列をベクトルに変換



# 前処理


画像
前処理
バッチ化
モデル

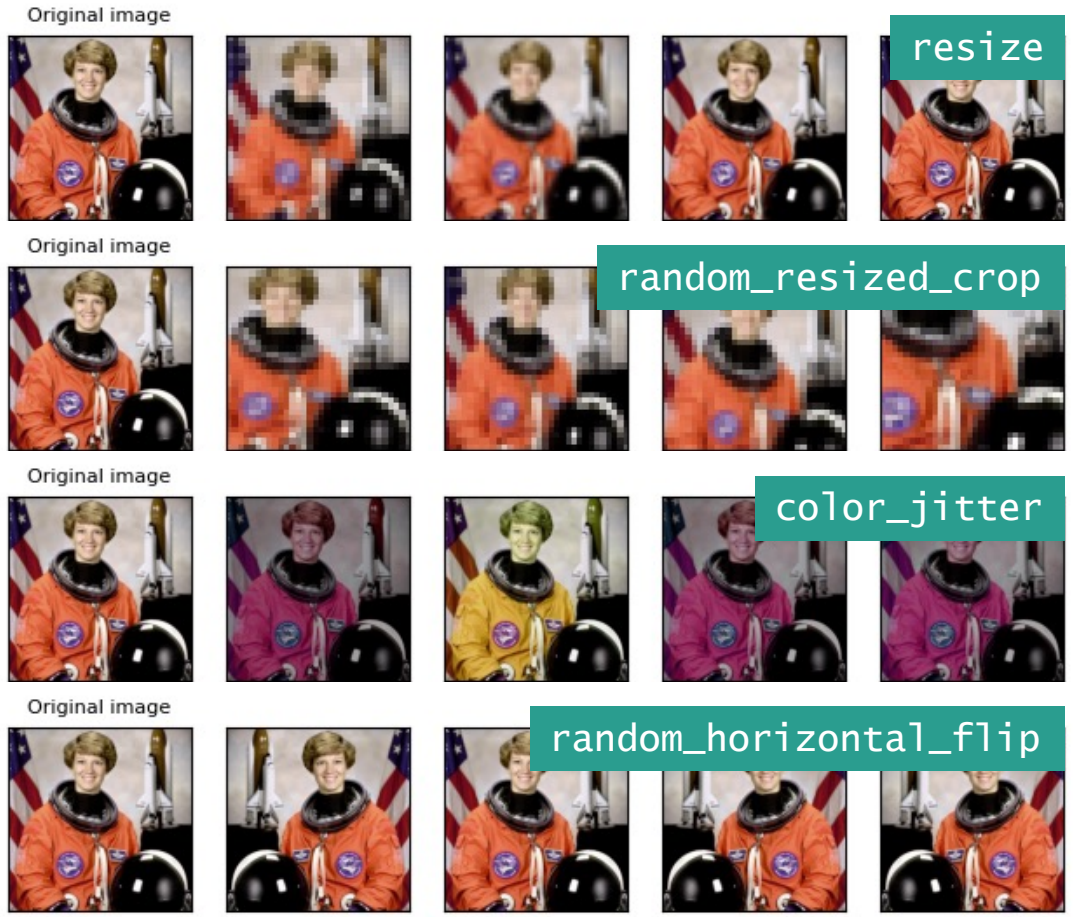

daisy
dandelion
roses
sunflowers
tulips


0 0 0 0 0 0
1 1 1 1 1 1
2 2 2 2 2 2
3 3 3 3 3 3
4 4 4 4 4 4


#1
3 4 3 1
#2
0 1 2 0
#3
4 2 4 0
#4
2 2 4 0

# 前処理


- 訓練画像を整理し、画像と教師ラベルの対応づけを行う、全訓練画像のリストを作成する。
- 画像に対する前処理を定義する。
  - 画像を行列データとして読み取る
  - サイズ変更、色調調整、アフィン変換
  - テンソルに変換
- リストの i 番目の画像がリクエストされたら、その画像を前処理して返す。



# 前処理


- 訓練画像を整理し、画像と教師ラベルの対応づけを行う、全訓練画像のリストを作成する。
- 画像に対する前処理を定義する。
  - 画像を行列データとして読み取る
  - サイズ変更、色調調整、アフィン変換
  - テンソルに変換
- リストの i 番目の画像がリクエストされたら、その画像を前処理して返す。

```
class Dataset:
  def __init__():
    self.x = images
    self.y = labels
    self.transform = transforms

  def __len__():
    retrun len(self.x)

  def __getitem__(i):
    x = self.transform(self.x[i])
    y = self.y[i]
    return x, y
```

```
class Resize:
  def __init__(x):
    self.x = x
  def __call__(x):
    x = resize(x)
    x
```

```
class Affine:
  def __init__(x):
    self.x = x
  def __call__(x):
    x = affine(x)
    x
```

```
class Tensor:
  def __init__(x):
    self.x = x
  def __call__(x):
    x = tensor(x)
    x
```

```
transforms = [
  Resize(),
  Affine(),
  Tensor(),
]
```

# バッチ化


- 画像リストを取得し、必要に応じて順番を調整し、バッチ化の処理を行う。

```
train_dataset = Dataset(images, labels)

train_dataloader = DataLoader(
  train_dataset,
  batch_size=16, shuffle=True, num_workers=2)
```

```
class Dataset:
  def __init__(images, labels):
    self.x = images
    self.y = labels
    self.transform = transforms

  def __len__():
    retrun len(self.x)

  def __getitem__(i):
    x = self.transform(self.x[i])
    y = self.y[i]
    return x, y
```

```
class Resize:
  def __init__(x):
    self.x = x
  def __call__(x):
    x = resize(x)
    x
```

```
class Affine:
  def __init__(x):
    self.x = x
  def __call__(x):
    x = affine(x)
    x
```

```
class Tensor:
  def __init__(x):
    self.x = x
  def __call__(x):
    x = tensor(x)
    x
```

```
transforms = [
  Resize(),
  Affine(),
  Tensor(),
]
```

# モデル訓練


- 小分けした画像データを少しずつモデルに代入し訓練を行う。

```
train_dataset = Dataset(images, labels)

train_dataloader = DataLoader(
  train_dataset,
  batch_size=16, shuffle=True, num_workers=2)
```

```
for inputs, labels in train_dataloder:
  optimizer.zero_grad()
  outputs = model(inputs)
  loss = criterion(outputs, inputs)
  loss.backward()
  optimizer.step()
```

```
class Dataset:
  def __init__(images, labels):
    self.x = images
    self.y = labels
    self.transform = transforms

  def __len__():
    retrun len(self.x)

  def __getitem__(i):
    x = self.transform(self.x[i])
    y = self.y[i]
    return x, y
```

```
class Resize:
  def
    se
  def
    x
    x
```

```
class Affine:
  def __init__(x):
    self.x = x
  def __call__(x):
    x = affine(x)
    x
```

```
class Tensor:
  def __init__(x):
    self.x = x
  def __call__(x):
    x = tensor(x)
    x
```

```
transforms = [
  Resize(),
  Affine(),
  Tensor(),
]
```

# モデル訓練

```
# dataset
train_dataset = Dataset(images, labels)
train_dataloader = DataLoader(train_dataset, batch_size=4)

# model
model = SimpleCNN()
model.train()

# parameters
criterion = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(net.parameters(), lr=0.001)
n_epochs = 50

# training
for epoch in range(n_epochs):
  running_loss = 0.0

  # mini batch
  for inputs, labels in train_dataloader:
    optimizer.zero_grad()
    outputs = model(inputs)
    loss = criterion(outputs, labels)
    loss.backward()
    optimizer.step()
    running_loss += loss.item()

  print('Epoch: {}; Loss: {}'.format(epoch, running_loss))
```

- モデルを訓練モードに切り替え、計算時に生じた勾配情報を保存する。

# モデル訓練

```
# dataset
train_dataset = Dataset(images, labels)
train_dataloader = DataLoader(train_dataset, batch_size=4)

# model
model = SimpleCNN()
model.train()

# parameters
criterion = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(net.parameters(), lr=0.001)
n_epochs = 50

# training
for epoch in range(n_epochs):
  running_loss = 0.0

  # mini batch
  for inputs, labels in train_dataloader:
    optimizer.zero_grad()
    outputs = model(inputs)
    loss = criterion(outputs, labels)
    loss.backward()
    optimizer.step()
    running_loss += loss.item()

  print('Epoch: {}; Loss: {}'.format(epoch, running_loss))
```

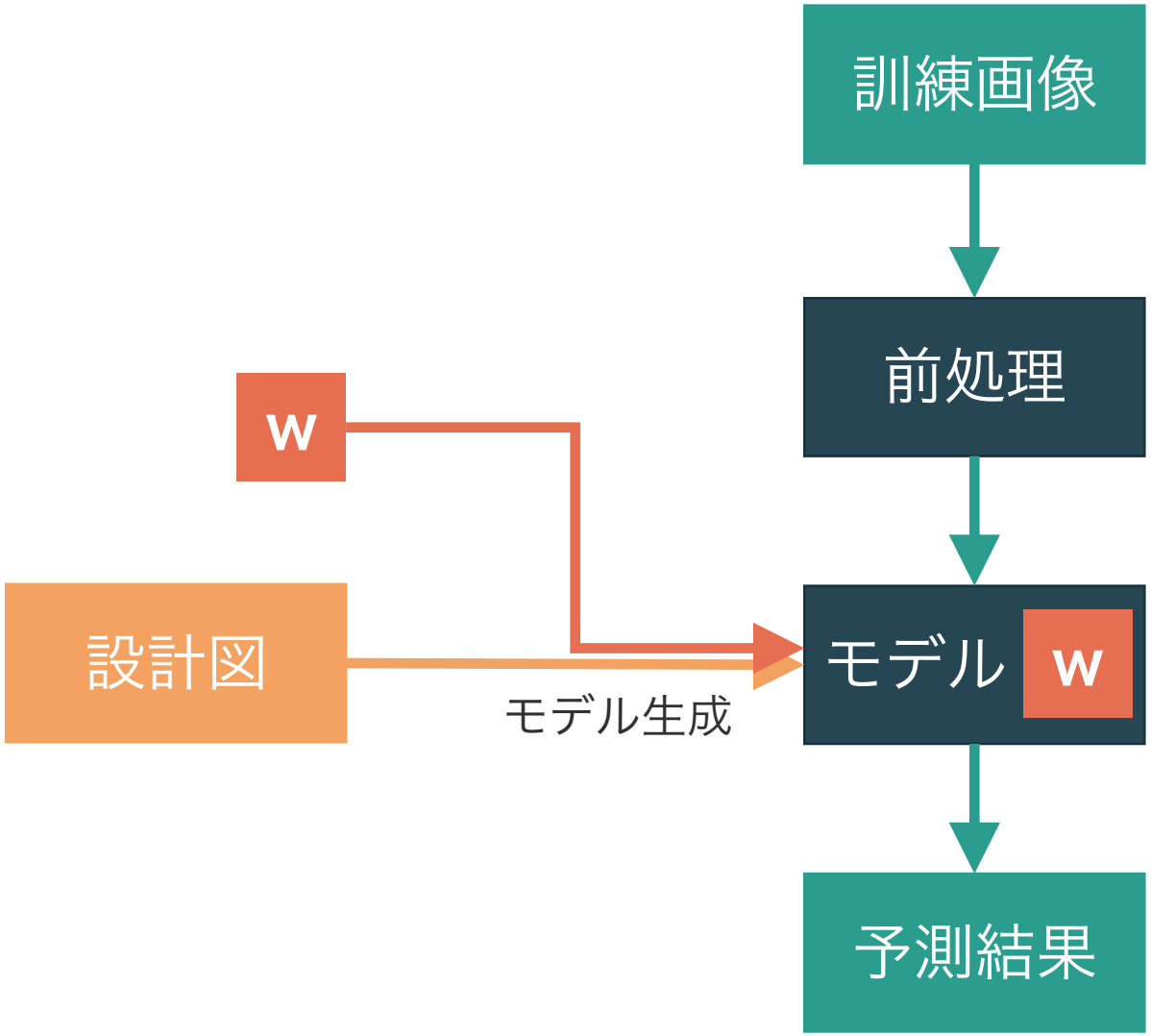
- 損失関数
  - 多クラス分類問題では一般的にクロスエントロピー関数を用いる。
  - 回帰分析では MSELoss などを使用する。
  - 独自で定義した関数を使用することもできる。その際、outputs と labels を受け取り、1 つの値を返す関数を定義すればよい。
- 最適化アルゴリズム
  - SGD や Adam などがよく使われる。
- エポック
  - 過学習を起こさない程度のエポックを設定。
  - 検証データを使用して early stopping を組み込む。

# モデル訓練

```
# dataset
train_dataset = Dataset(images, labels)
train_dataloader = DataLoader(train_dataset, batch_size=4)

# model
model = SimpleCNN()
model.train()

# parameters
criterion = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(net.parameters(), lr=0.001)
n_epochs = 50

# training
for epoch in range(n_epochs):
  running_loss = 0.0

  # mini batch
  for inputs, labels in train_dataloader:
    optimizer.zero_grad()                  # 1
    outputs = model(inputs)                # 2
    loss = criterion(outputs, labels)      # 3
    loss.backward()                        # 4
    optimizer.step()                       # 5
    running_loss += loss.item()

  print('Epoch: {}; Loss: {}'.format(epoch, running_loss))
```

1. 古い損失（勾配情報）を消去する。
2. 画像データをモデルに代入し予測する。
3. 予測値と教師ラベルから損失（誤差）を計算する。
4. 損失をネットワーク全体に逆伝播する。
5. 逆伝播された損失を利用してパラメータを更新する。

# 推論




```
# model
model = SimpleCNN()
model.load_state_dict(torch.load('weight.pth'))
model.eval()
```

```
# preprocess an image
img_fpath = '/path/to/image.jpg'
im = PIL.Image.open(img_fpath)
im = ImageOps.exif_transpose(im)
im = inference_transform(im)
im = im.unsqueeze(0)
```

```
# inference
output = model(input)
# [[0.0935, -5.4933, -1.1156, 6.0684, 0.0409]]

output_softmax = torch.softmax(output, dim=1)
# [[0.0025, 0.0000, 0.0001, 0.9950, 0.0023]]

output_sigmoid = torch.softmax(output)
# [[0.5233, 0.0041, 0.2468, 0.9977, 0.5102]]
```

# モデル構造

```
from torch.nn as nn
from torchvision import models

model = models.vgg16(pretrained=True)

print(model)
```

classifier[5] の出力数 4096 を受け取る。

```
model.classifier[6] = nn.Linear(4096, 5)
```

出力数を 5 に設定する。

```
VGG(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU(inplace=True)
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (6): ReLU(inplace=True)
    (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (8): ReLU(inplace=True)
    (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (13): ReLU(inplace=True)
    (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (15): ReLU(inplace=True)
    (16): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (17): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (18): ReLU(inplace=True)
    (19): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (20): ReLU(inplace=True)
    (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (22): ReLU(inplace=True)
    (23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (24): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (25): ReLU(inplace=True)
    (26): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (27): ReLU(inplace=True)
    (28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (29): ReLU(inplace=True)
    (30): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(7, 7))
  (classifier): Sequential(
    (0): Linear(in_features=25088, out_features=4096, bias=True)
    (1): ReLU(inplace=True)
    (2): Dropout(p=0.5, inplace=False)
    (3): Linear(in_features=4096, out_features=4096, bias=True)
    (4): ReLU(inplace=True)
    (5): Dropout(p=0.5, inplace=False)
    (6): Linear(in_features=4096, out_features=1000, bias=True)
  )
)
```

# モデル構造

```
from torch.nn as nn
from torchvision import models

model = models.resnet18(pretrained=True)

print(model)
```

avgpool の出力数 512 を受け取る。

```
model.fc = nn.Linear(512, 5)
```

出力数を 5 に設定する。

```
ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
  (layer1): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (1): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (layer2): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)

    (1): BasicBlock(
      (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
  (fc): Linear(in_features=512, out_features=1000, bias=True)
)
```

# モデル構造

```
from torch.nn as nn
from torchvision import models

model = models.vgg16(pretrained=True)
model.classifier[6] = nn.Linear(4096, 5)


for param in model.features.parameters():
  param.requires_grad = False
```

```
VGG(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU(inplace=True)
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (6): ReLU(inplace=True)
    (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (8): ReLU(inplace=True)
    (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (13): ReLU(inplace=True)
    (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (15): ReLU(inplace=True)
    (16): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (17): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (18): ReLU(inplace=True)
    (19): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (20): ReLU(inplace=True)
    (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (22): ReLU(inplace=True)
    (23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (24): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (25): ReLU(inplace=True)
    (26): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (27): ReLU(inplace=True)
    (28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (29): ReLU(inplace=True)
    (30): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(7, 7))
  (classifier): Sequential(
    (0): Linear(in_features=25088, out_features=4096, bias=True)
    (1): ReLU(inplace=True)
    (2): Dropout(p=0.5, inplace=False)
    (3): Linear(in_features=4096, out_features=4096, bias=True)
    (4): ReLU(inplace=True)
    (5): Dropout(p=0.5, inplace=False)
    (6): Linear(in_features=4096, out_features=1000, bias=True)
  )
)
```

# モデル構造

```
from torch.nn as nn
from torchvision import models

model = models.vgg16(pretrained=True)
model.classifier[6] = nn.Linear(4096, 5)


for param in model.features.parameters():
  param.requires_grad = False

for param in model.avgpool.parameters():
  param.requires_grad = False
```

```
VGG(
 (features): Sequential(
   (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
   (1): ReLU(inplace=True)
   (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
   (3): ReLU(inplace=True)
   (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
   (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
   (6): ReLU(inplace=True)
   (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
   (8): ReLU(inplace=True)
   (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
   (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
   (11): ReLU(inplace=True)
   (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
   (13): ReLU(inplace=True)
   (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
   (15): ReLU(inplace=True)
   (16): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
   (17): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
   (18): ReLU(inplace=True)
   (19): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
   (20): ReLU(inplace=True)
   (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
   (22): ReLU(inplace=True)
   (23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
   (24): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
   (25): ReLU(inplace=True)
   (26): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
   (27): ReLU(inplace=True)
   (28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
   (29): ReLU(inplace=True)
   (30): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
 )
 (avgpool): AdaptiveAvgPool2d(output_size=(7, 7))
 (classifier): Sequential(
   (0): Linear(in_features=25088, out_features=4096, bias=True)
   (1): ReLU(inplace=True)
   (2): Dropout(p=0.5, inplace=False)
   (3): Linear(in_features=4096, out_features=4096, bias=True)
   (4): ReLU(inplace=True)
   (5): Dropout(p=0.5, inplace=False)
   (6): Linear(in_features=4096, out_features=1000, bias=True)
 )
)
```

# モデル構造

```
from torch.nn as nn
from torchvision import models

model = models.vgg16(pretrained=True)
model.classifier[6] = nn.Linear(4096, 5)


for i, param in \
    enumerate(model.features.parameters()):

  param.requires_grad = False

  if i > 15:
    break
```

```
VGG(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU(inplace=True)
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (6): ReLU(inplace=True)
    (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (8): ReLU(inplace=True)
    (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (13): ReLU(inplace=True)
    (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (15): ReLU(inplace=True)
    (16): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (17): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (18): ReLU(inplace=True)
    (19): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (20): ReLU(inplace=True)
    (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (22): ReLU(inplace=True)
    (23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (24): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (25): ReLU(inplace=True)
    (26): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (27): ReLU(inplace=True)
    (28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (29): ReLU(inplace=True)
    (30): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(7, 7))
  (classifier): Sequential(
    (0): Linear(in_features=25088, out_features=4096, bias=True)
    (1): ReLU(inplace=True)
    (2): Dropout(p=0.5, inplace=False)
    (3): Linear(in_features=4096, out_features=4096, bias=True)
    (4): ReLU(inplace=True)
    (5): Dropout(p=0.5, inplace=False)
    (6): Linear(in_features=4096, out_features=1000, bias=True)
  )
)
```

# 物体検出入門

物体分類 – torch

**物体検出アルゴリズム**

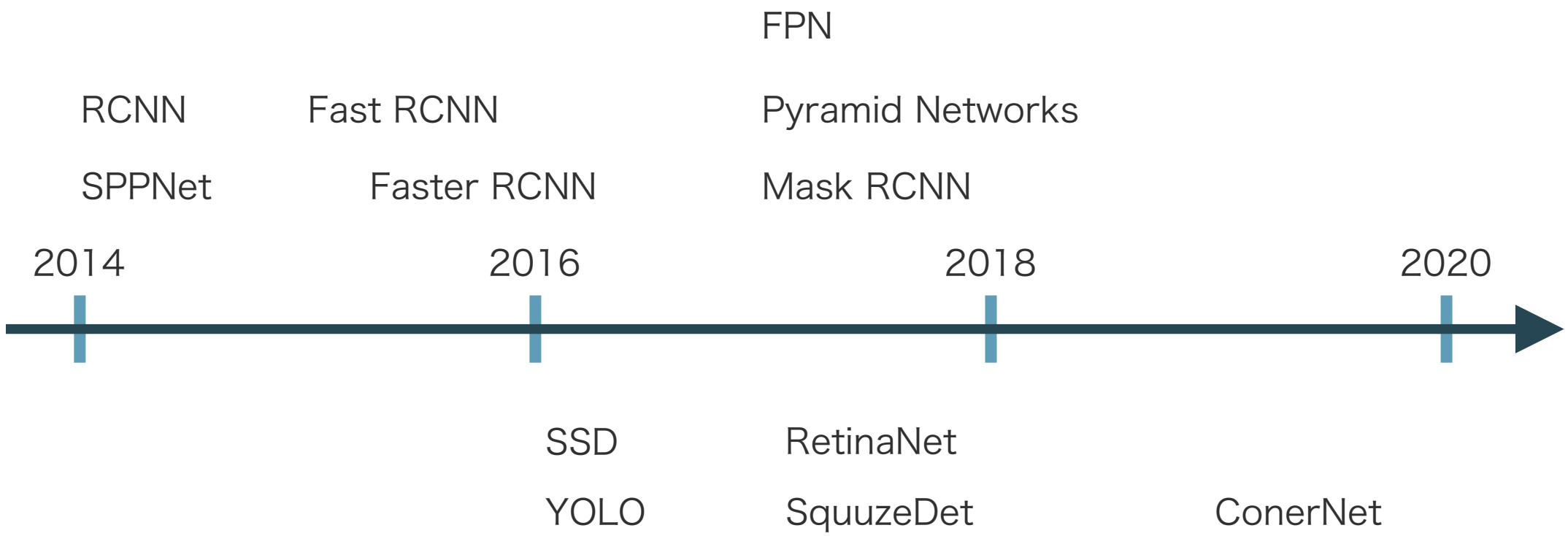
アノテーションツール

物体検出 – Detectron2

物体検出 – MMDetection

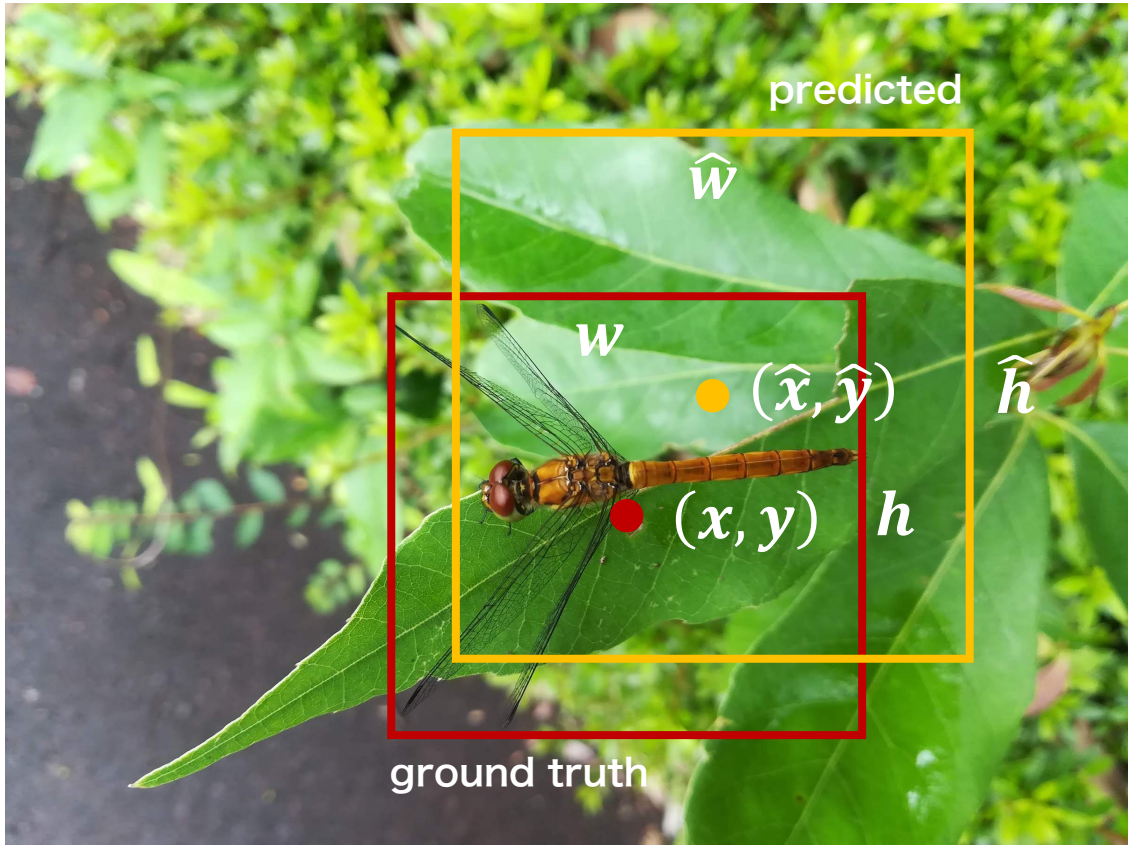
# 物体検出アルゴリズム

**Two-stage methods**



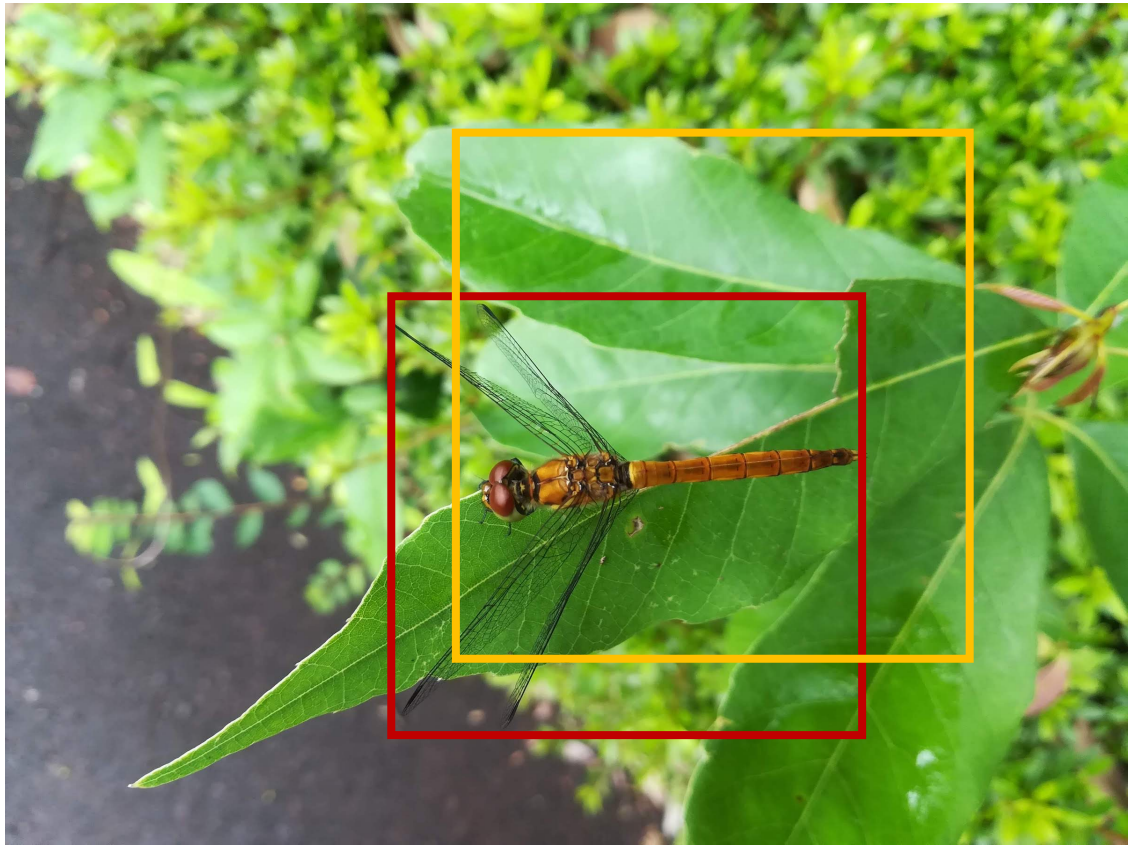

**One-stage methods**

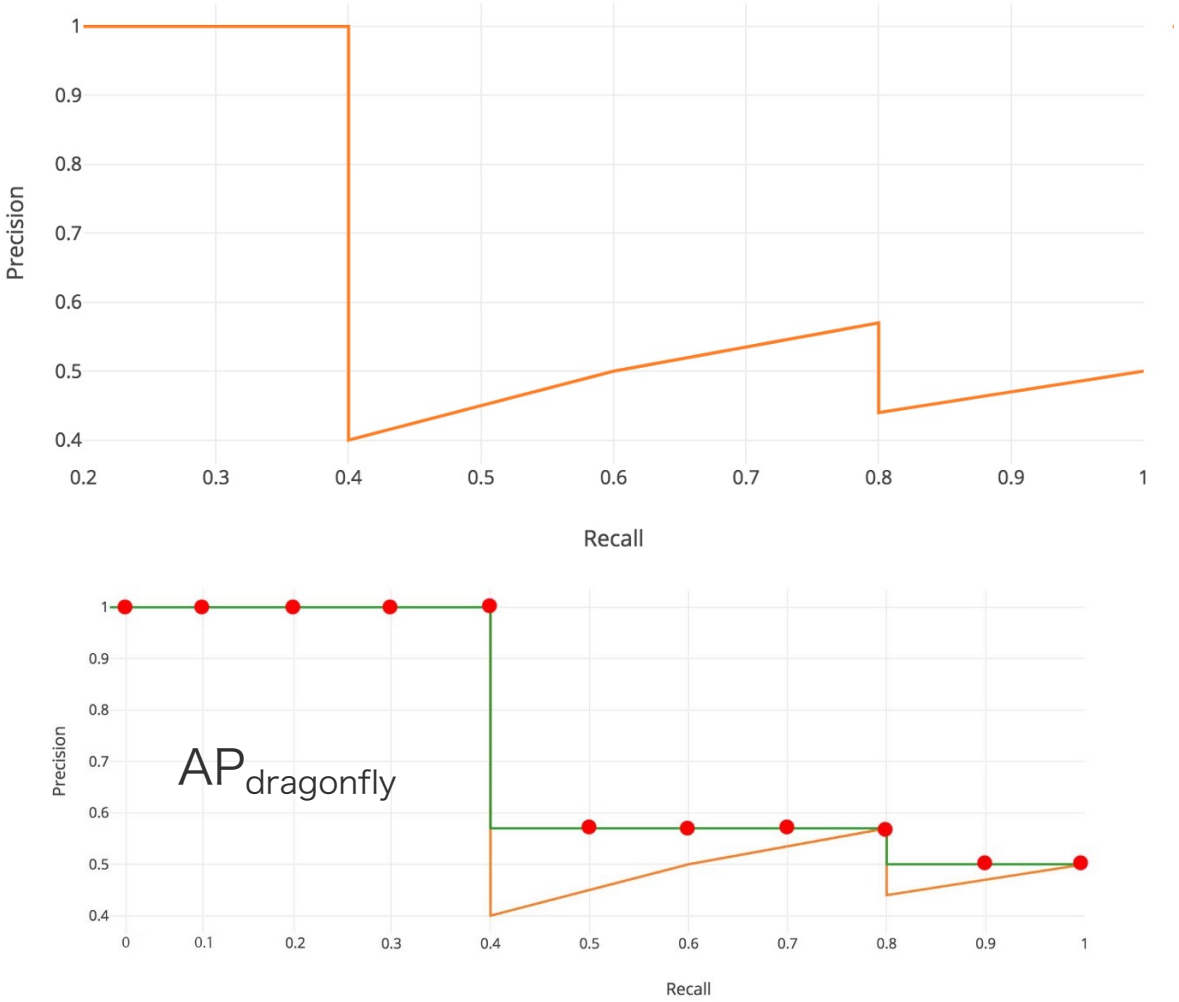
# Intersection over Union (IoU)




$$\text{IoU} = \frac{\text{intersection}}{\text{union}}$$

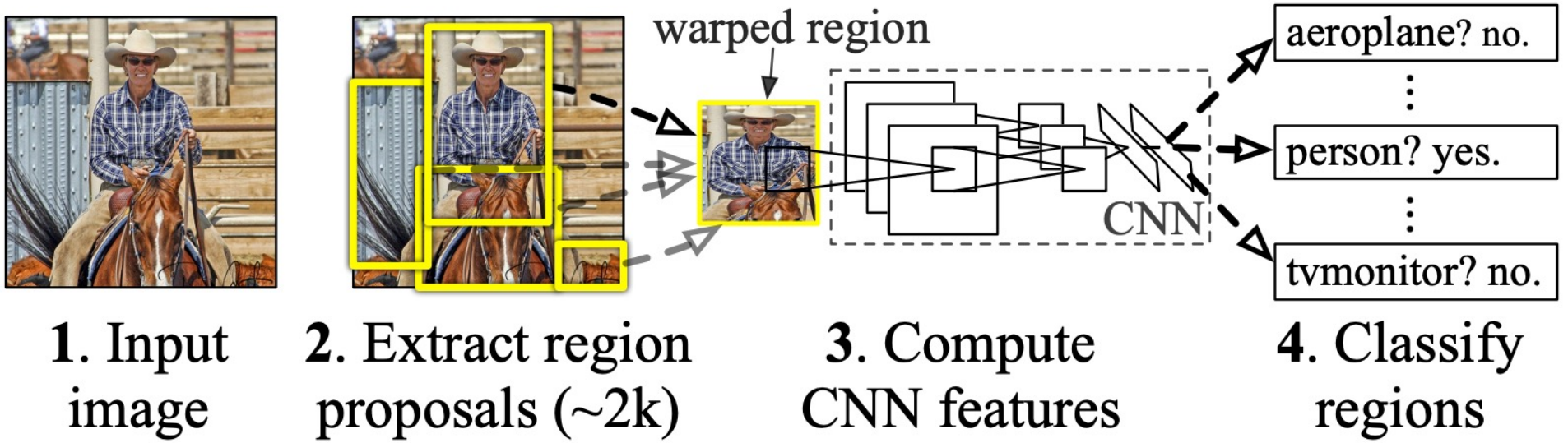
# mean Average Precision (mAP)






https://jonathan-hui.medium.com/map-mean-average-precision-for-object-detection-45c121a31173

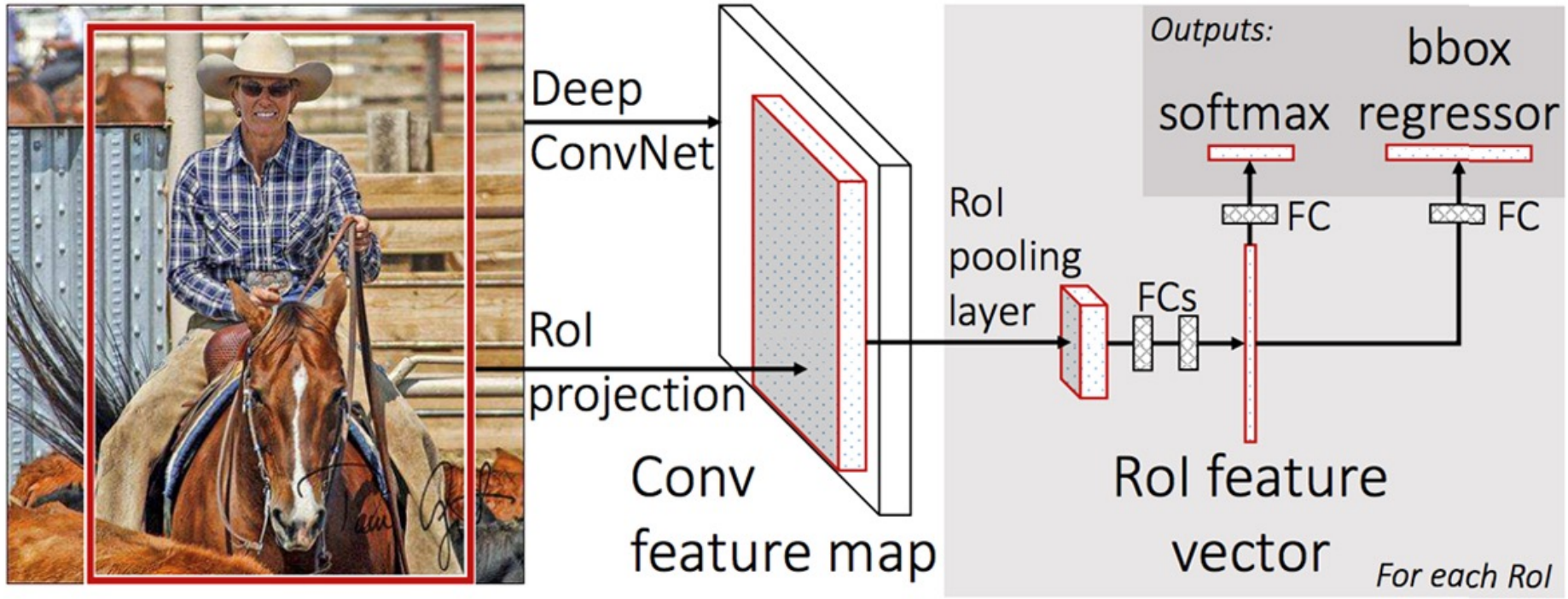
# R-CNN




- select search アルゴリズムを利用し、入力画像にあるオブジェクトが存在していそうな領域候補（RoI）を決定する。
- 各候補領域を CNN に入力し特徴抽出を行う。
- CNN から出力される特徴を SVM に入力し分類を行う。

https://arxiv.org/abs/1311.2524

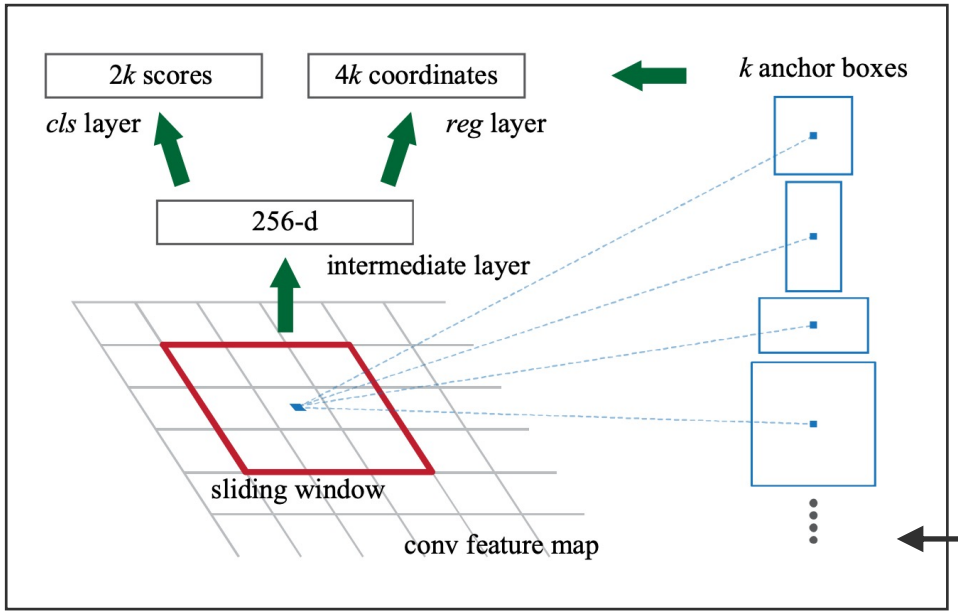
# Fast R-CNN

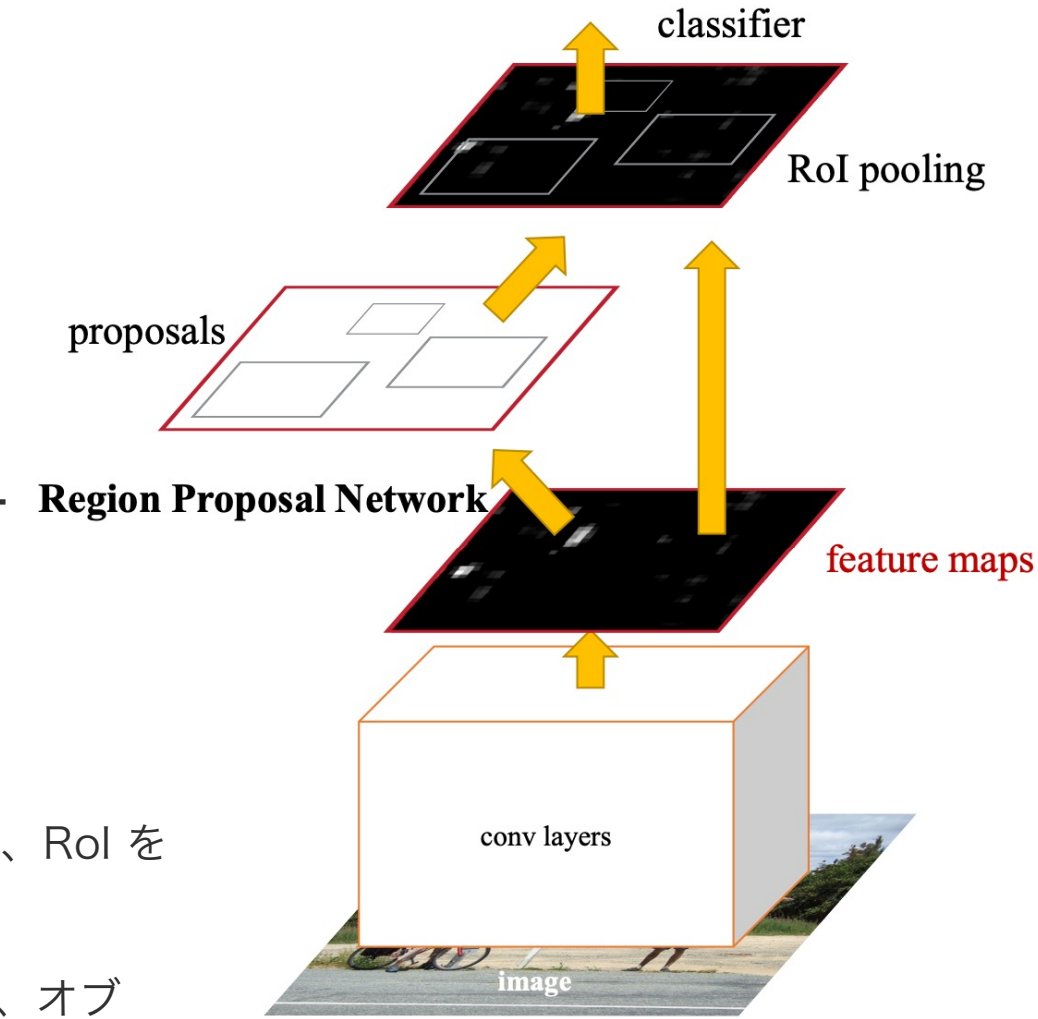



- 入力画像を CNN に入力し特徴マップを計算する。
- 入力画像をもとに決定した RoI を特徴マップに射影する。
- 特徴マップ上の RoI を処理し、物体分類とバウンディングボックスの座標を予測する。

https://arxiv.org/abs/1504.08083

# Faster R-CNN

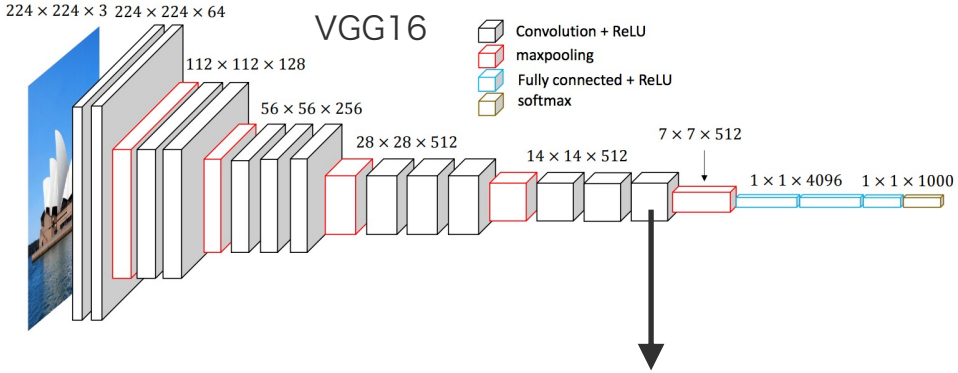



- 入力画像を CNN に入力し特徴マップを計算する。
- 特徴マップを Region Proposal Network に入力し、RoI を決定する。
- 特徴マップと RPN から出力される RoI を利用して、オブジェクトの分類を行う。

https://arxiv.org/abs/1506.01497

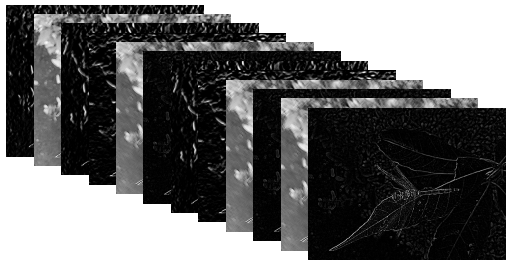
# Resion Proposal Network



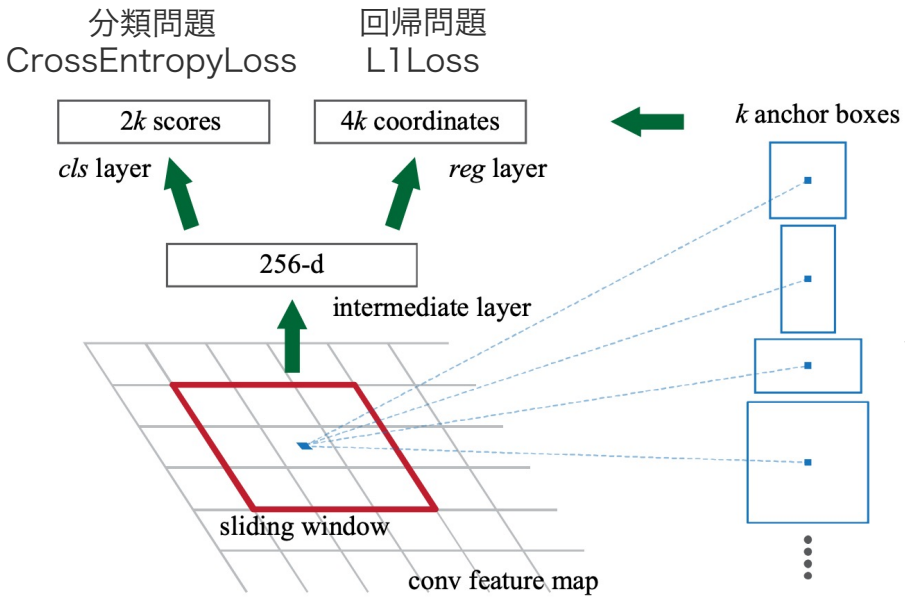

input

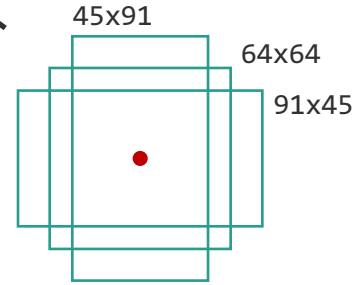
1024x768x3

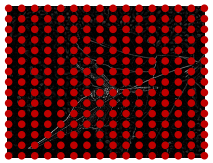
feature map



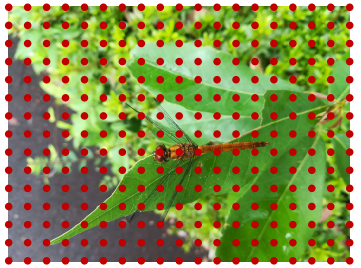
64x48x512



64x48 anchors



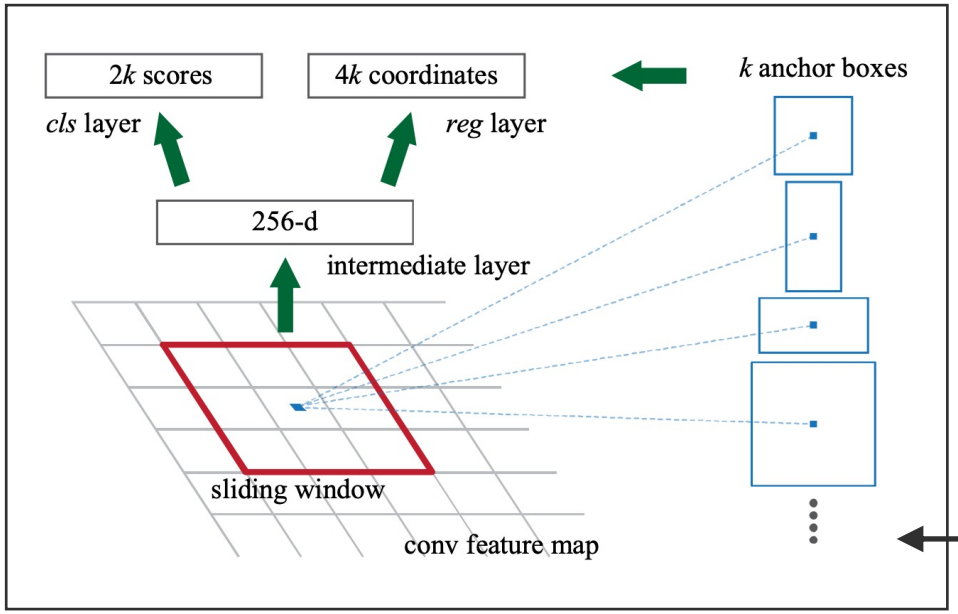
64x48 anchors

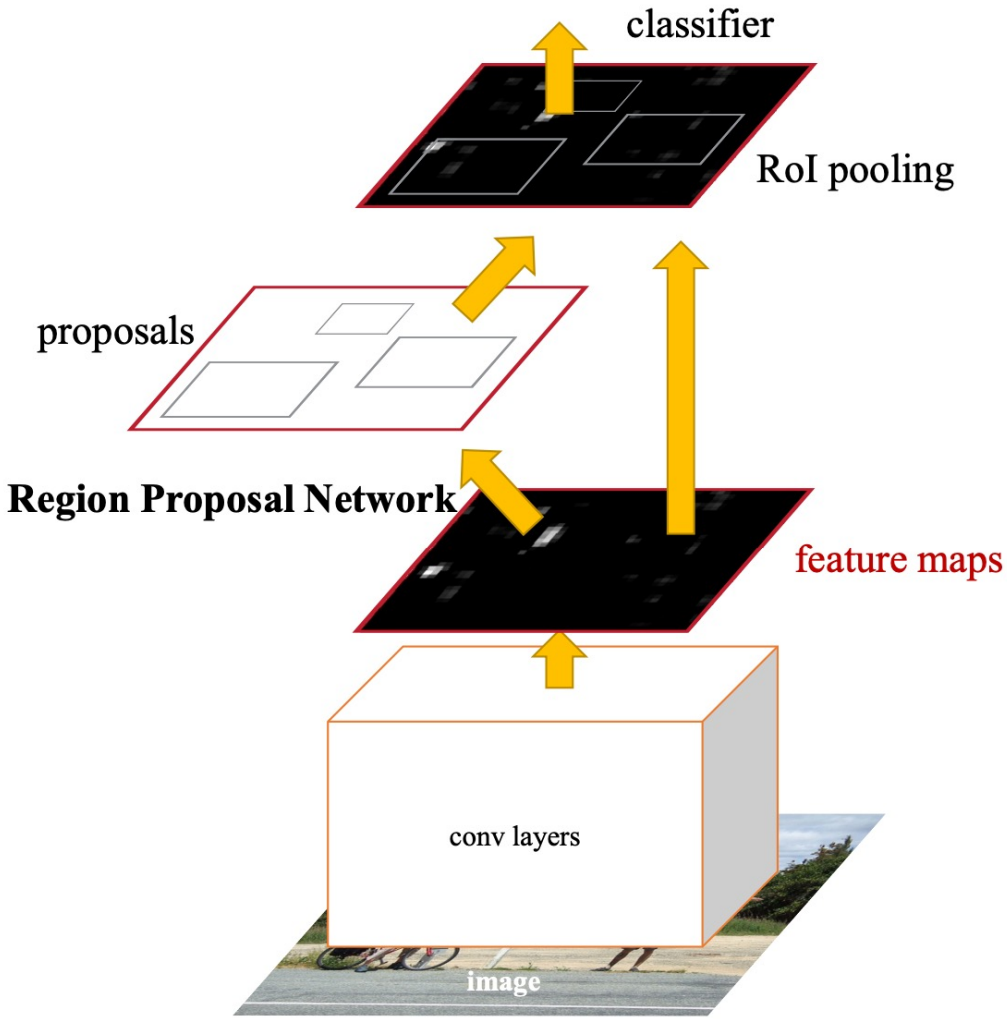
9 anchor boxes / anchor

parameters
  base width: 64, 128, 256
  ratio: 1:1, 1:2, 2:1

https://arxiv.org/abs/1506.01497
https://doi.org/10.1145/3038912.3052638

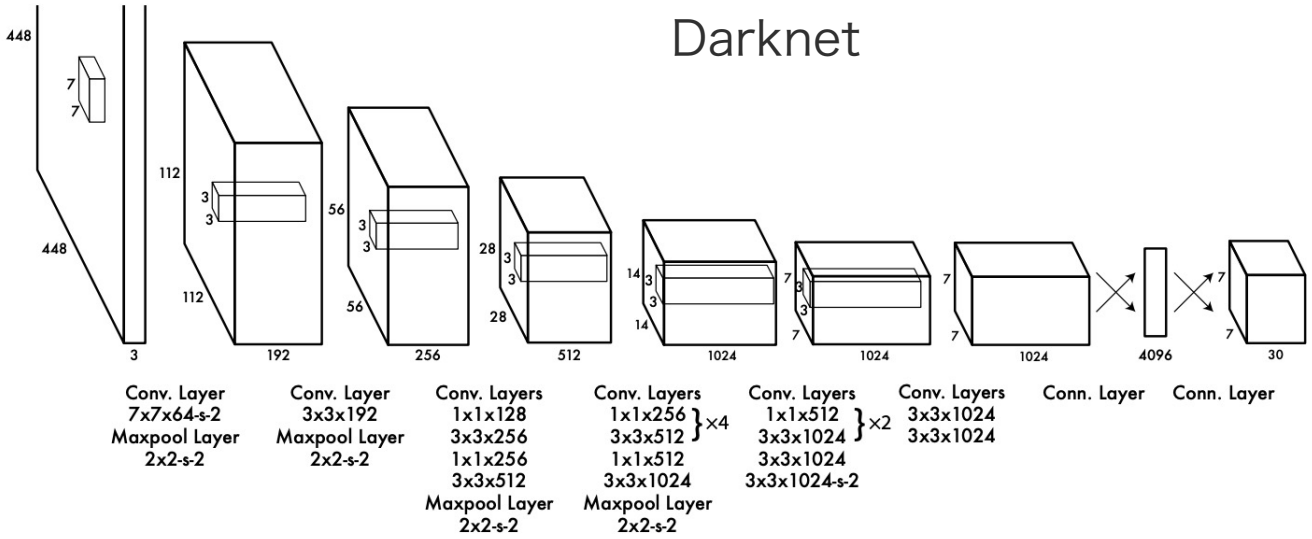
# Faster R-CNN

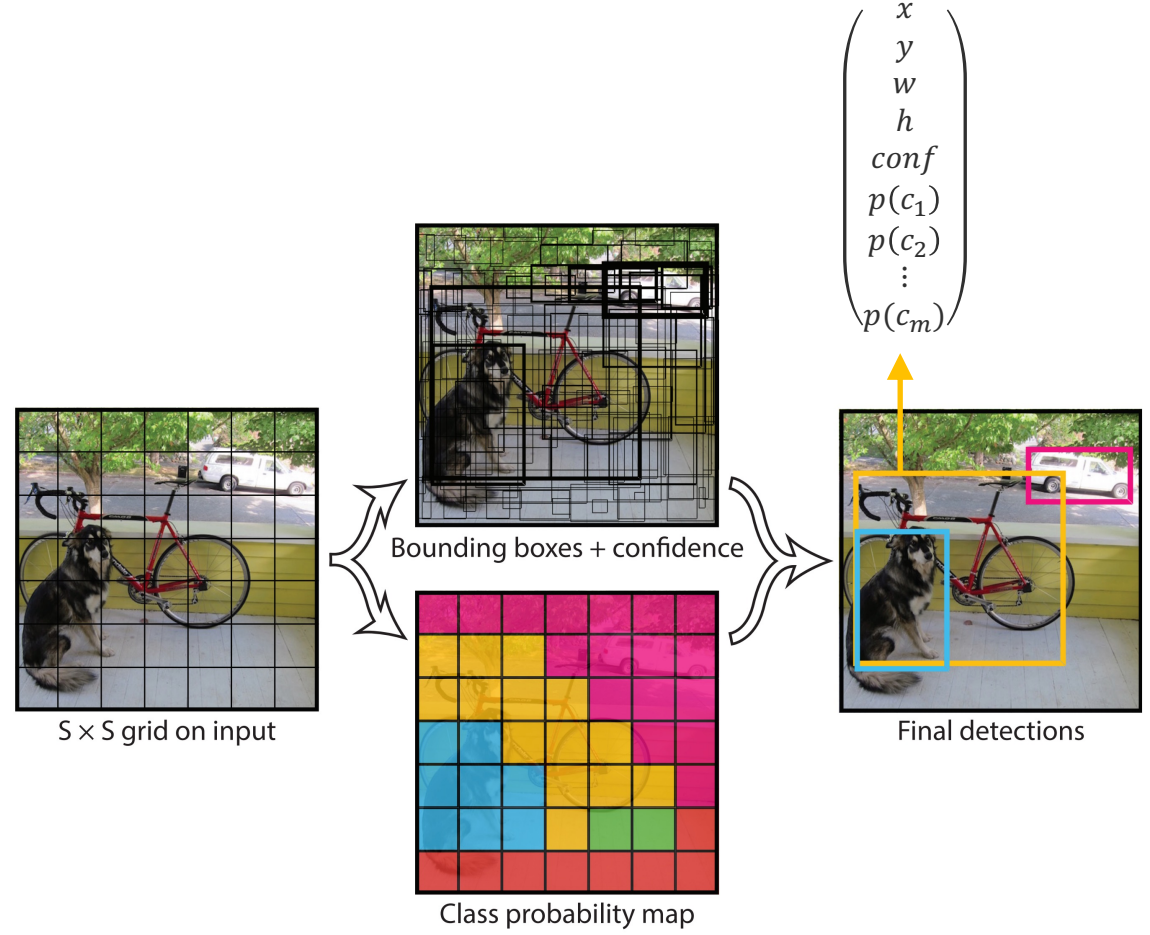



https://arxiv.org/abs/1506.01497

# You Only Look Once (YOLO)

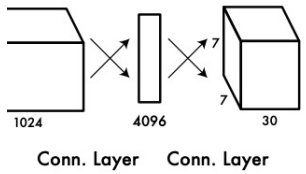


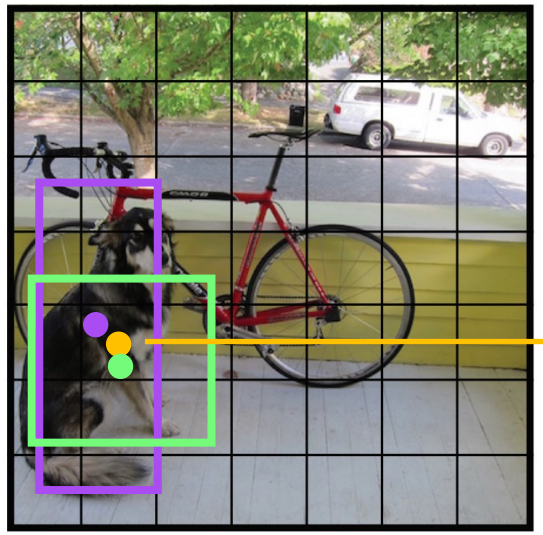




https://arxiv.org/abs/1506.02640

# YOLO – モデル訓練

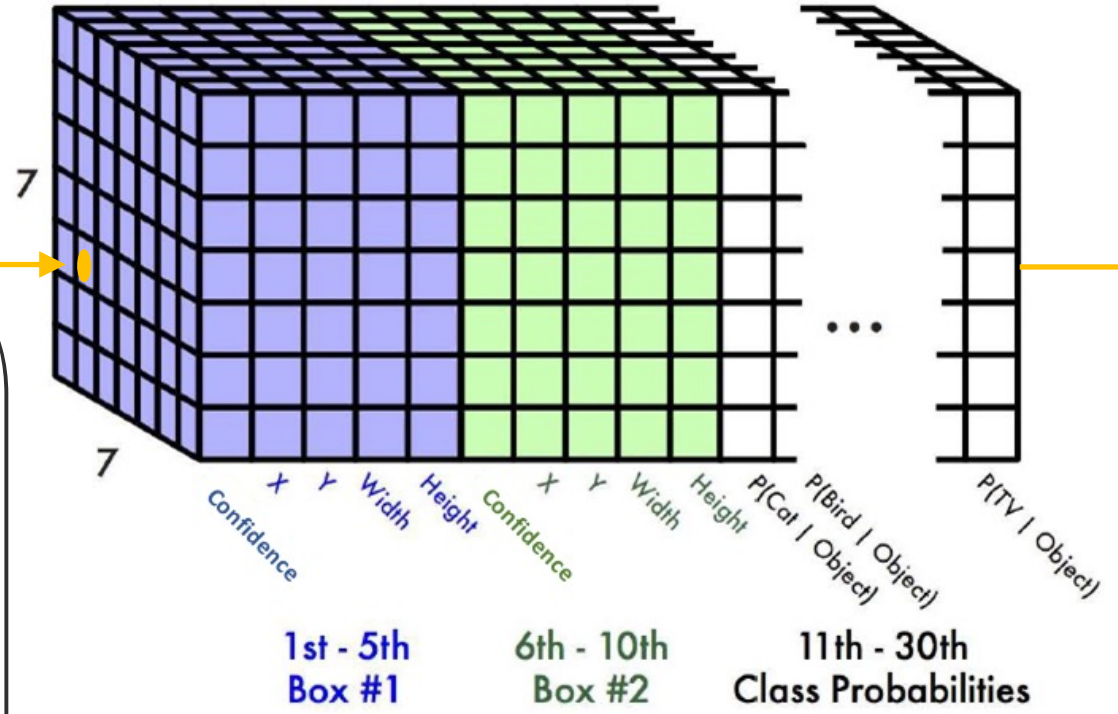
$$conf = \underset{\hat{b}\in\{b_1,b_2\}}{\text{argmax}}\{p(object)\cdot \text{IoU}_{\hat{b}}^{truth}\}$$



S × S grid on input

$$\begin{pmatrix} x \\ y \\ w \\ h \\ conf \\ x \\ y \\ w \\ h \\ conf \\ p(c_1) \\ p(c_2) \\ \vdots \\ p(c_m) \end{pmatrix}$$




$$\begin{pmatrix} 0.286 \\ 0.714 \\ 0.332 \\ 0.584 \\ 0.845 \\ 0 \\ 1 \\ \vdots \\ 0 \end{pmatrix} \quad p(\text{dog})$$

教師ラベル

https://arxiv.org/abs/1506.02640

# YOLO – 損失関数

1: オブジェクト
0: 背景

bbox 中心座標の予測誤差

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[ (x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right]$$

bbox の幅と高さの予測誤差

$$+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[ \left( \sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left( \sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right]$$

confidence の予測誤差

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left( C_i - \hat{C}_i \right)^2$$

自信を持って生成した bbox が偶然に教師ラベルと重なった場合、損失がゼロになる。

confidence の予測誤差（背景セルならば $C_i = 0$）

$$+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left( C_i - \hat{C}_i \right)^2$$

自信を持って生成した bbox にオブジェクトが含まれていなければ、その分だけ損失を計上する。

クラスの予測誤差

$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} \left( p_i(c) - \hat{p}_i(c) \right)^2$$

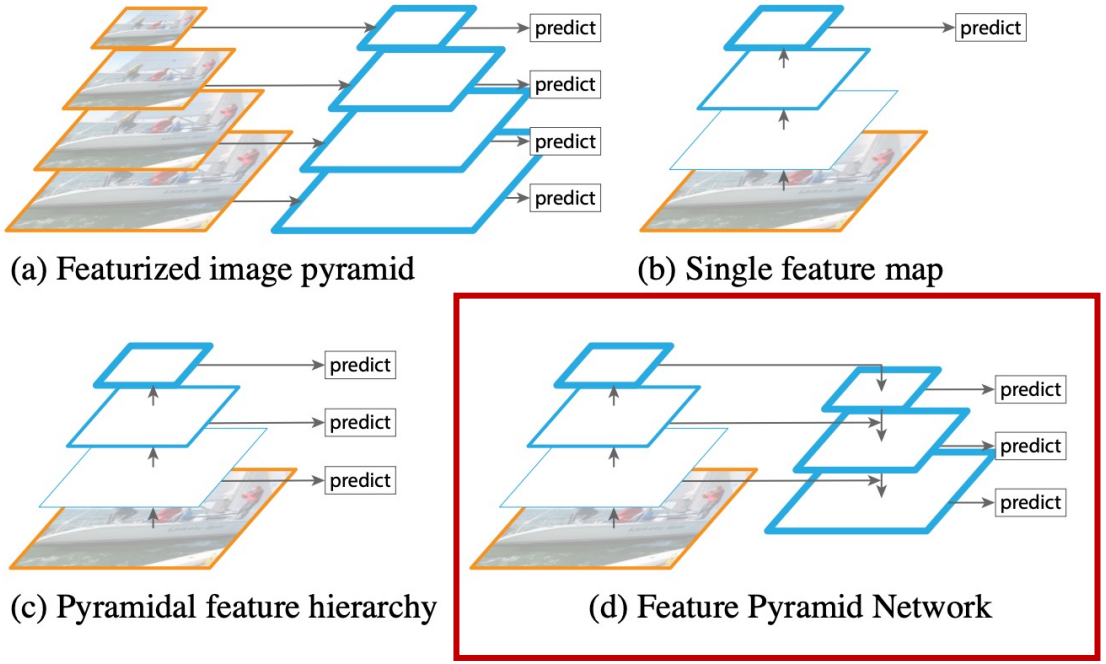
訓練過程で、オブジェクトのあるセルではバウンディングボックスの中心位置および幅と高さが学習され、信頼度も増加する。背景のセルでは学習されずに出鱈目なバウンディングボックスを低信頼度で生成される。

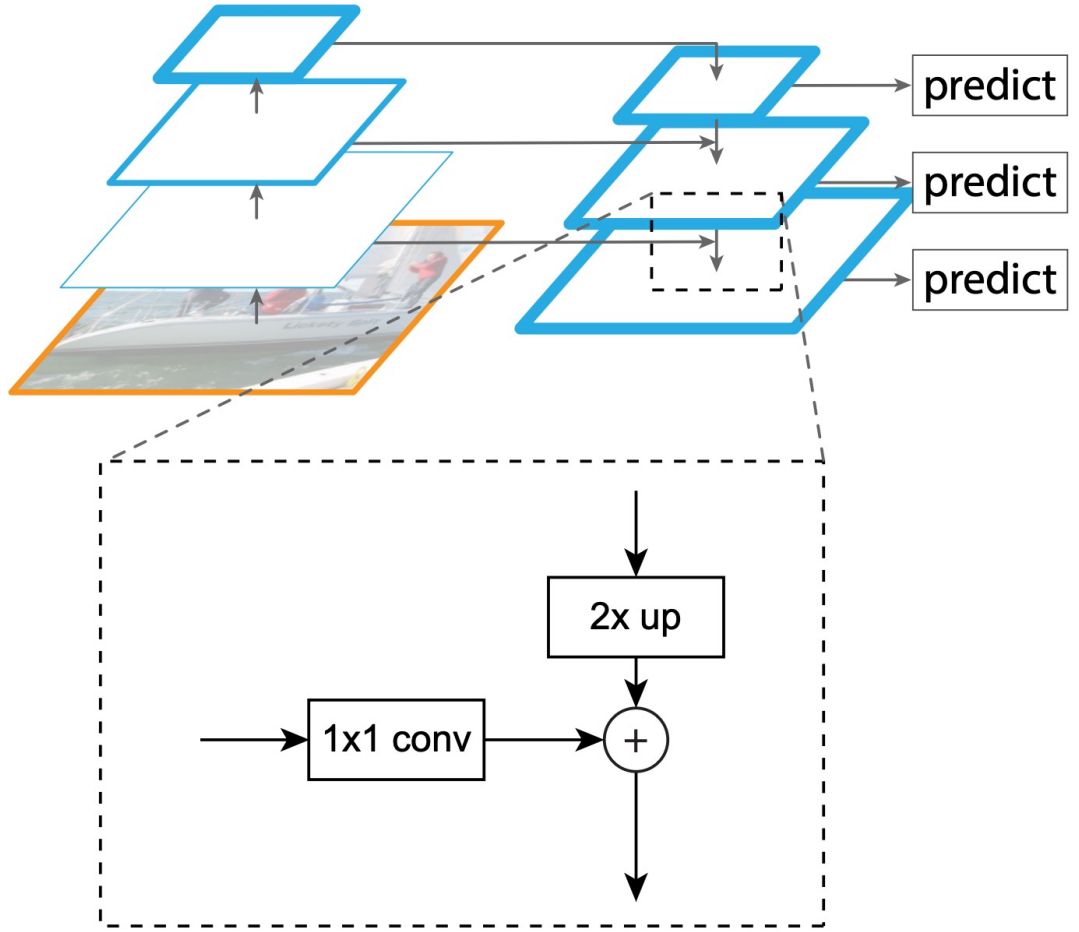
予測誤差が非常に小さいので、定数倍 $\lambda_{\text{coord}} = 5$ をかけて大きくする。

画像の大部分が背景であるため、この項が大きな値を取りやすく、他の項の値が相対的に小さくなり、学習が進まない。この効果を弱めるために、定数倍 $\lambda_{\text{noobj}} = 0.5$ をかける。

# YOLO

# Feature Pyramid Network (FPN)



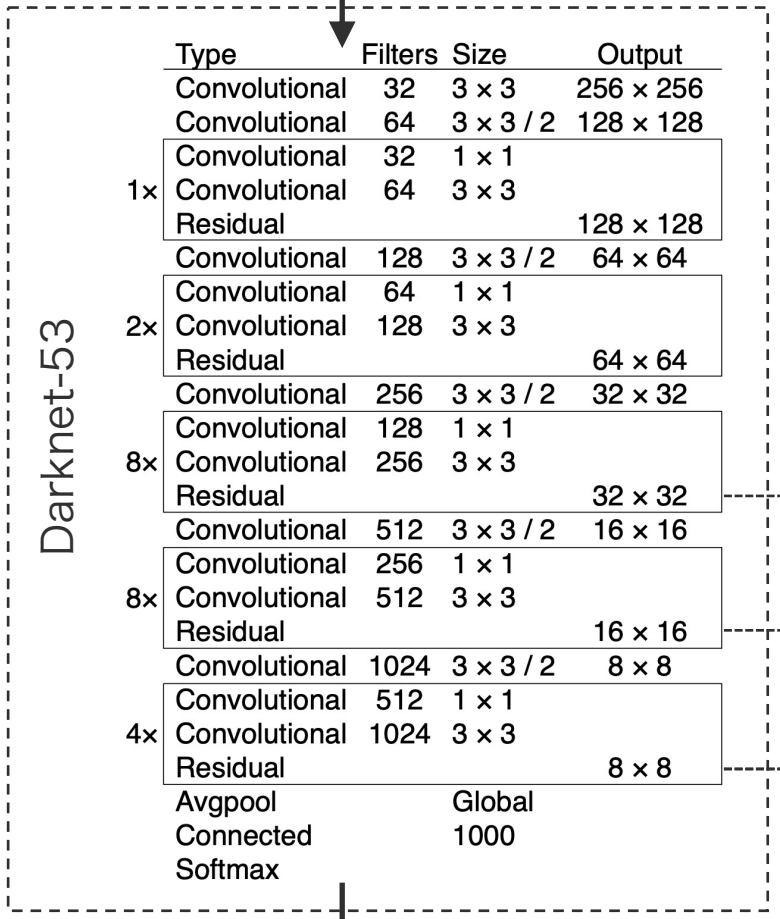



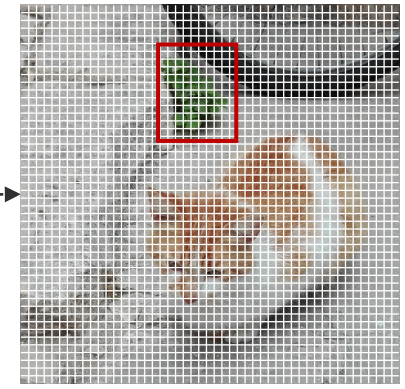

https://arxiv.org/abs/1612.03144

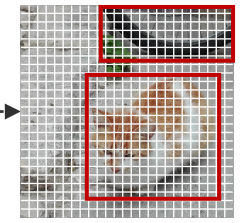
# YOLO3

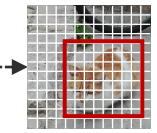
Darknet-53

| | Type | Filters | Size | Output |
|---|---|---|---|---|
| | Convolutional | 32 | 3 × 3 | 256 × 256 |
| | Convolutional | 64 | 3 × 3 / 2 | 128 × 128 |
| 1× | Convolutional | 32 | 1 × 1 | |
| | Convolutional | 64 | 3 × 3 | |
| | Residual | | | 128 × 128 |
| | Convolutional | 128 | 3 × 3 / 2 | 64 × 64 |
| 2× | Convolutional | 64 | 1 × 1 | |
| | Convolutional | 128 | 3 × 3 | |
| | Residual | | | 64 × 64 |
| | Convolutional | 256 | 3 × 3 / 2 | 32 × 32 |
| 8× | Convolutional | 128 | 1 × 1 | |
| | Convolutional | 256 | 3 × 3 | |
| | Residual | | | 32 × 32 |
| | Convolutional | 512 | 3 × 3 / 2 | 16 × 16 |
| 8× | Convolutional | 256 | 1 × 1 | |
| | Convolutional | 512 | 3 × 3 | |
| | Residual | | | 16 × 16 |
| | Convolutional | 1024 | 3 × 3 / 2 | 8 × 8 |
| 4× | Convolutional | 512 | 1 × 1 | |
| | Convolutional | 1024 | 3 × 3 | |
| | Residual | | | 8 × 8 |
| | Avgpool | | Global | |
| | Connected | | 1000 | |
| | Softmax | | | |





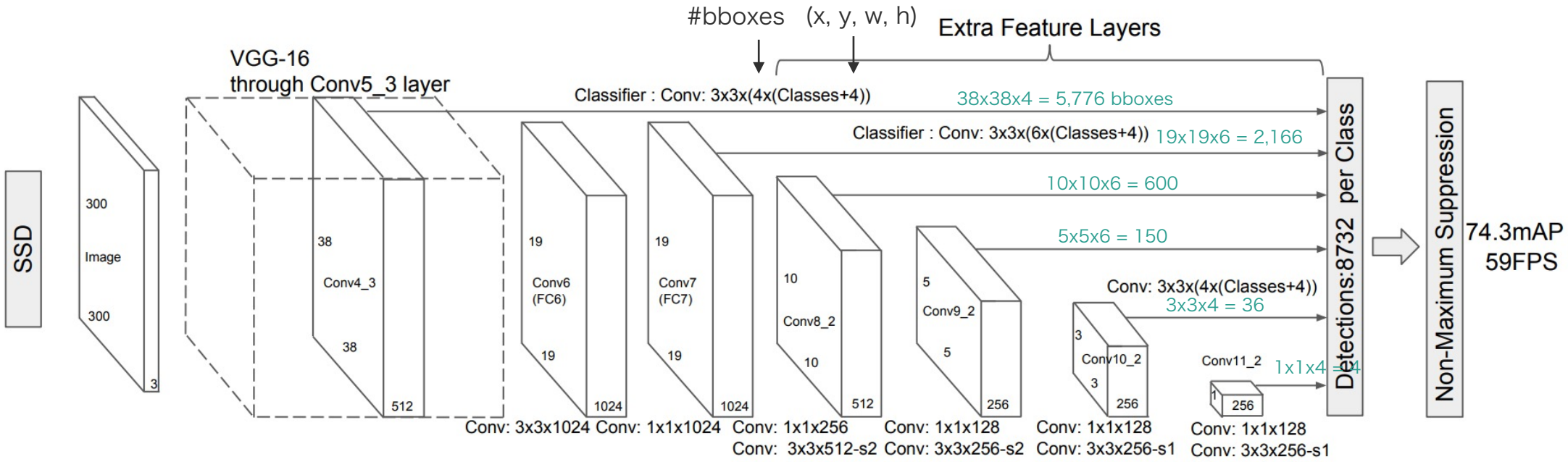


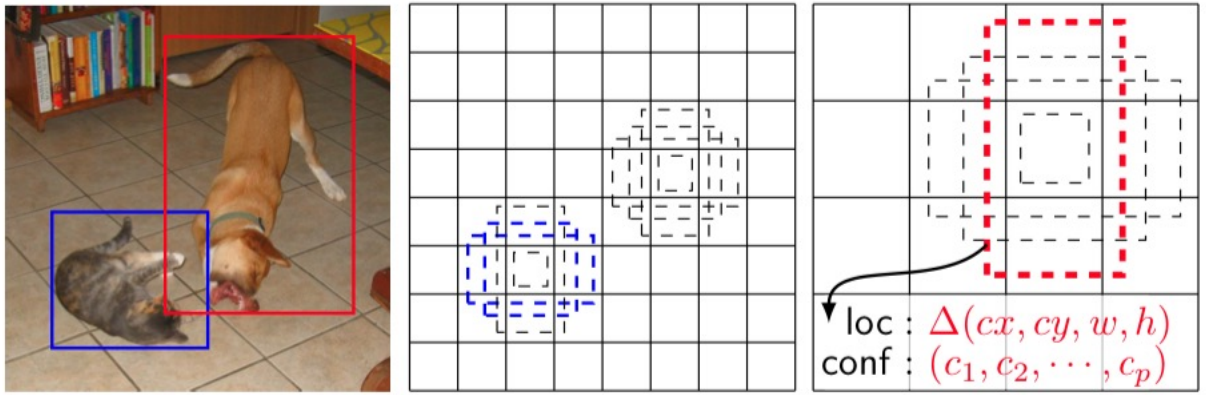
small objects

large objects

https://arxiv.org/abs/1804.02767

# SSD (Single Shot multibox Detector)

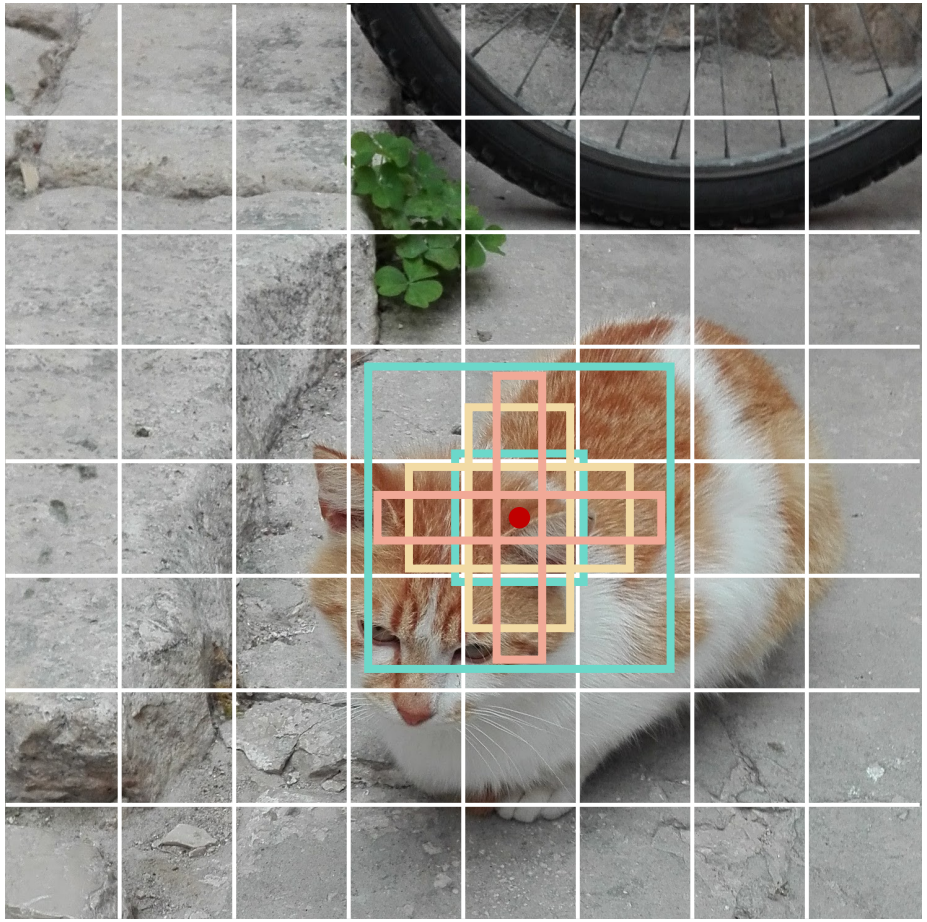


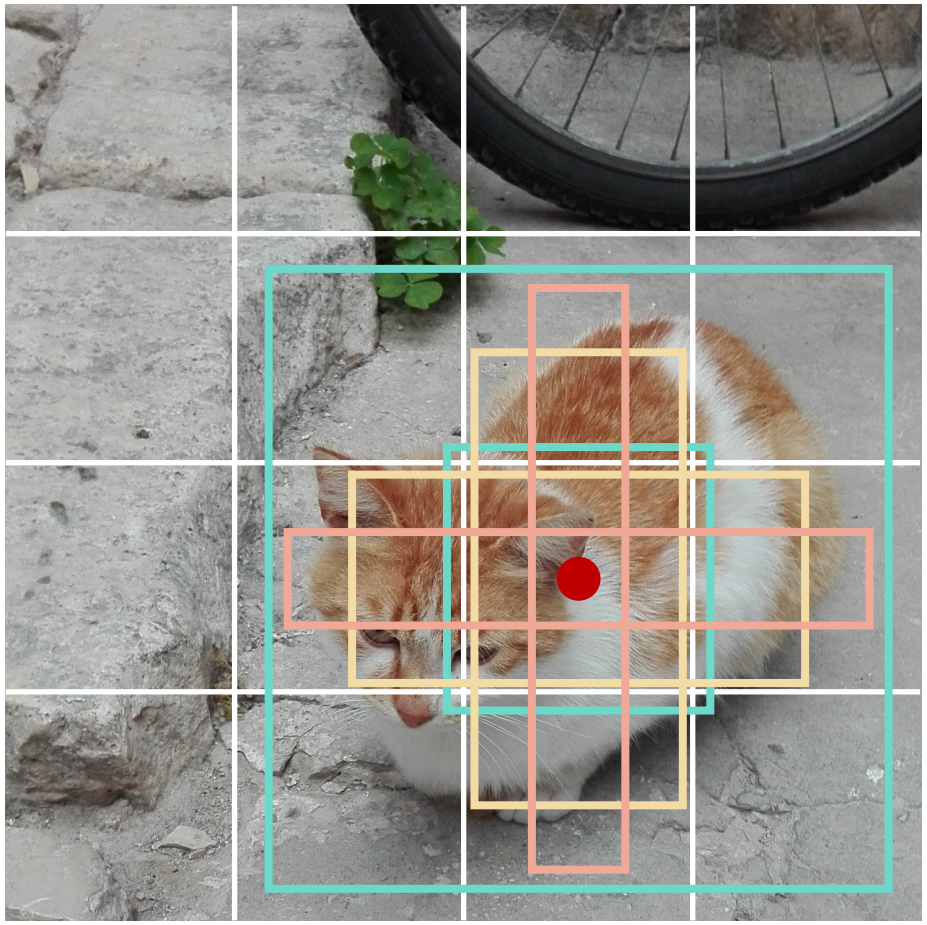




(a) Image with GT boxes (b) $8 \times 8$ feature map (c) $4 \times 4$ feature map

https://arxiv.org/abs/1512.02325

# SSD – バウンディングボックス比率





$(s_m, s_m)$　$(\sqrt{s_m s_M}, \sqrt{s_m s_M})$

$(s_k\sqrt{2}, s_k/\sqrt{2})$　$(s_k/\sqrt{2}, s_k\sqrt{2})$

$(s_k\sqrt{3}, s_k/\sqrt{3})$　$(s_k/\sqrt{3}, s_k\sqrt{3})$

# SSD – 損失関数

$$L(x, c, l, g) = \frac{1}{N}(L_{conf}(x, c) + \alpha L_{loc}(x, l, g))$$

クラスの予測誤差

bbox の予測誤差

IoU > 0.5 の positive matches のみに対して損失を計上する。数多くの negative matches は無視される。

$$L_{loc}(x, l, g) = \sum_{i \in Pos}^{N} \sum_{m \in \{cx, cy, w, h\}} x_{ij}^{k} \text{smooth}_{\text{L1}}(l_i^m - \hat{g}_j^m)$$

$$\hat{g}_j^{cx} = (g_j^{cx} - d_i^{cx})/d_i^w \qquad \hat{g}_j^{cy} = (g_j^{cy} - d_i^{cy})/d_i^h$$

$$\hat{g}_j^w = \log\left(\frac{g_j^w}{d_i^w}\right) \qquad \hat{g}_j^h = \log\left(\frac{g_j^h}{d_i^h}\right)$$

class $p$

$$L_{conf}(x, c) = -\sum_{i \in Pos}^{N} x_{ij}^{p} log(\hat{c}_i^p) - \sum_{i \in Neg} log(\hat{c}_i^0) \quad \text{where} \quad \hat{c}_i^p = \frac{\exp(c_i^p)}{\sum_p \exp(c_i^p)}$$

$i^{th}$ predicted bbox

$j^{th}$ ground truth bbox

positive matches でクラスを正しく予測できたならば損失を減らす。negative matches で背景を正しく予測できたならば損失を減らす。

$$x_{ij}^p = f(x) = \begin{cases} 1, & \text{IoU} > 0.5 \\ 0, & \text{IoU} \leq 0.5 \end{cases}$$

https://arxiv.org/abs/1512.02325

# RetinaNet

## Cross Entropy Loss

$$\mathrm{CE}(p, y) = \begin{cases} -\log(p) & \text{if } y = 1 \\ -\log(1-p) & \text{otherwise.} \end{cases}$$

推定確率 $p_t$ 次のように定義する

$$p_t = \begin{cases} p & \text{if } y = 1 \\ 1-p & \text{otherwise,} \end{cases}$$

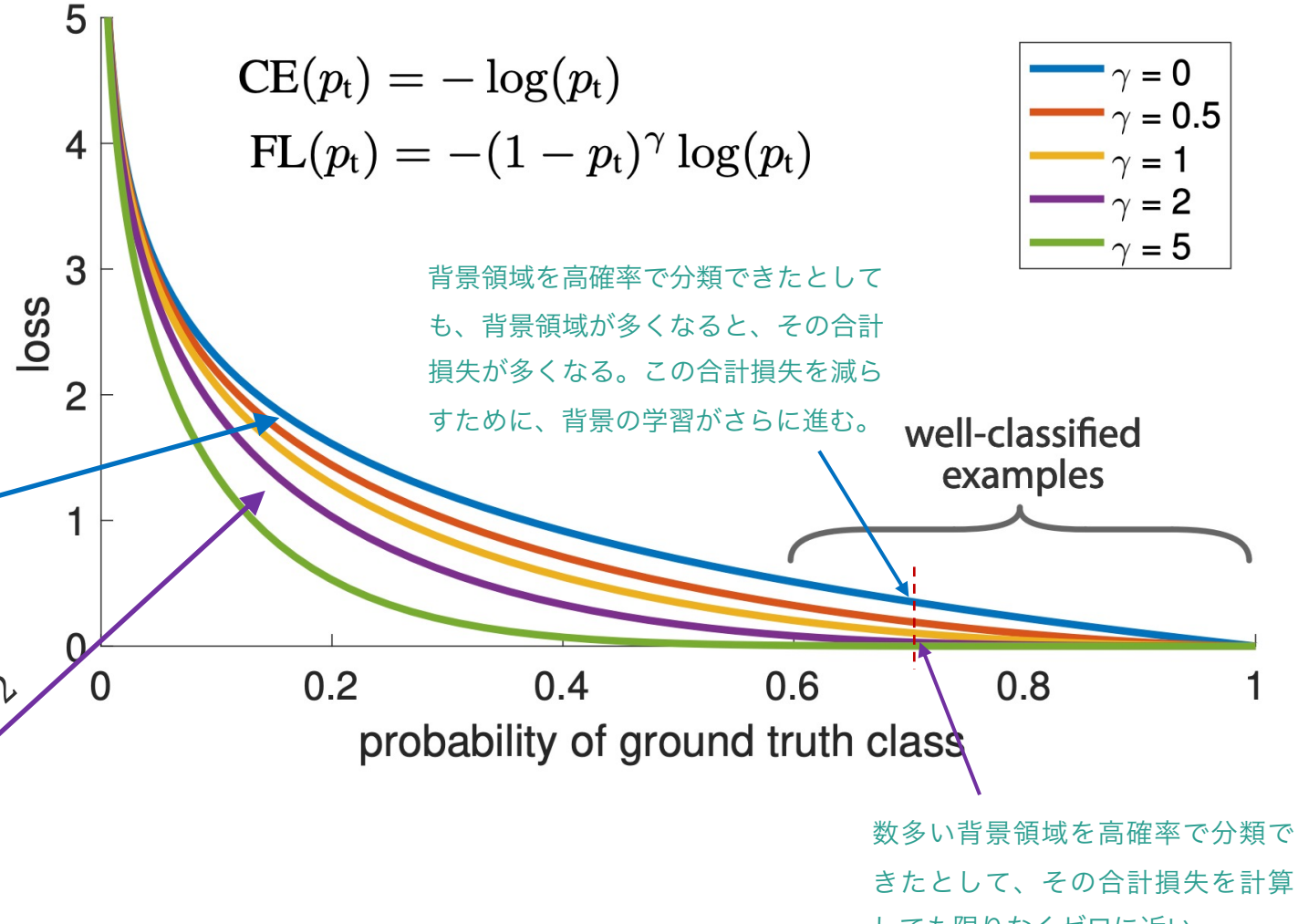
$$\mathrm{CE}(p, y) = \mathrm{CE}(p_t) = -\log(p_t)$$

## Focal Loss

$$\mathrm{FL}(p_t) = -\alpha_t(1-p_t)^\gamma \log(p_t).$$

$$\mathrm{CE}(p_t) = -\log(p_t)$$

$$\mathrm{FL}(p_t) = -(1-p_t)^\gamma \log(p_t)$$

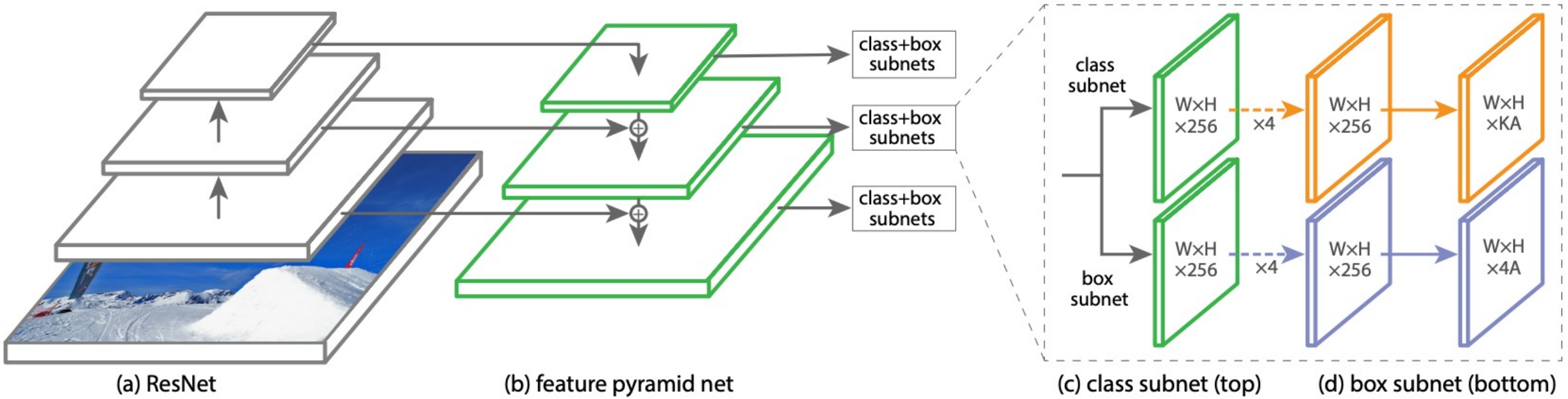



背景領域を高確率で分類できたとしても、背景領域が多くなると、その合計損失が多くなる。この合計損失を減らすために、背景の学習がさらに進む。

数多い背景領域を高確率で分類できたとして、その合計損失を計算しても限りなくゼロに近い。

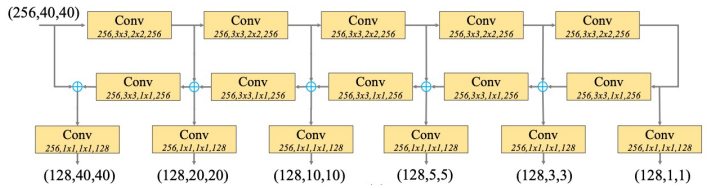
https://arxiv.org/abs/1708.02002

# RetinaNet

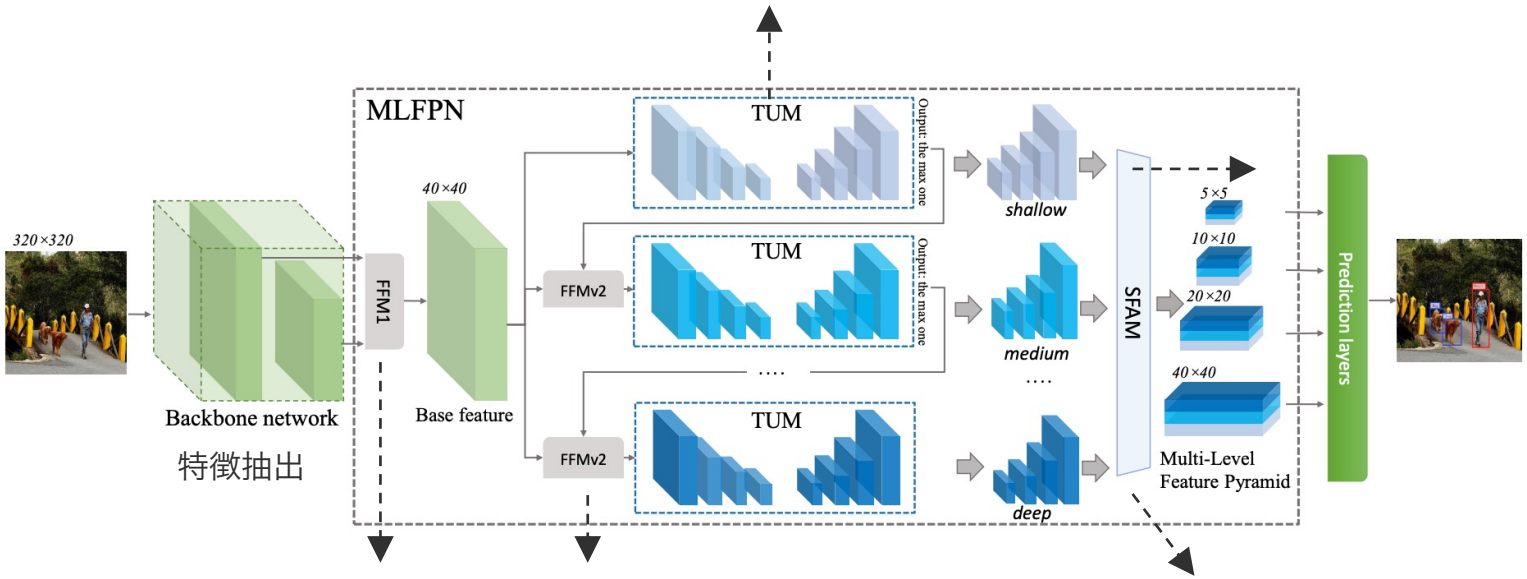



https://arxiv.org/abs/1708.02002

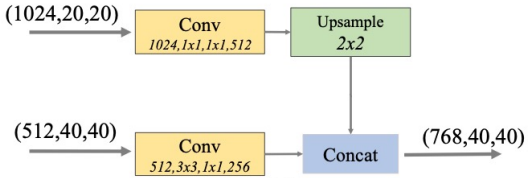
# M2Det



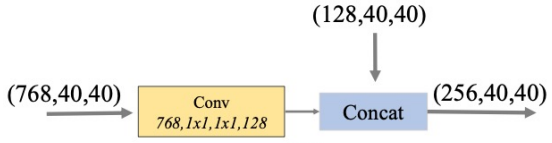

Thinned U-shape Module (TUM)



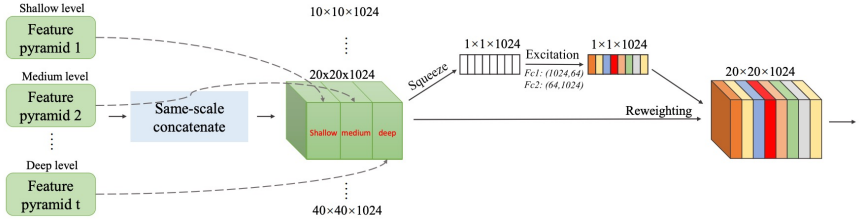

Feature Fusion Module v1 (FFMv1)




Feature Fusion Module v2 (FFMv2)

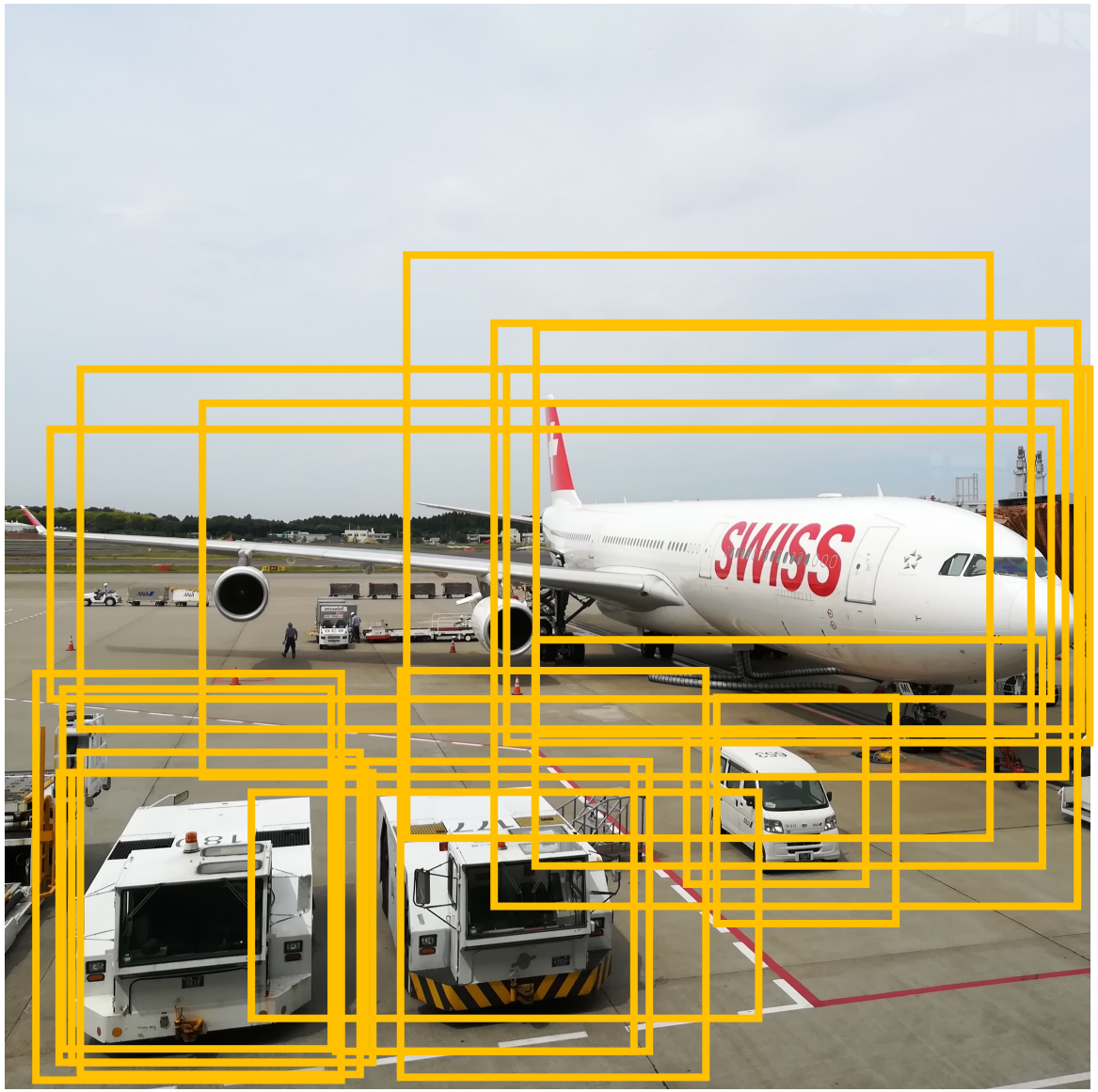



Scale-wise Feature Aggregation Module (SFAM)



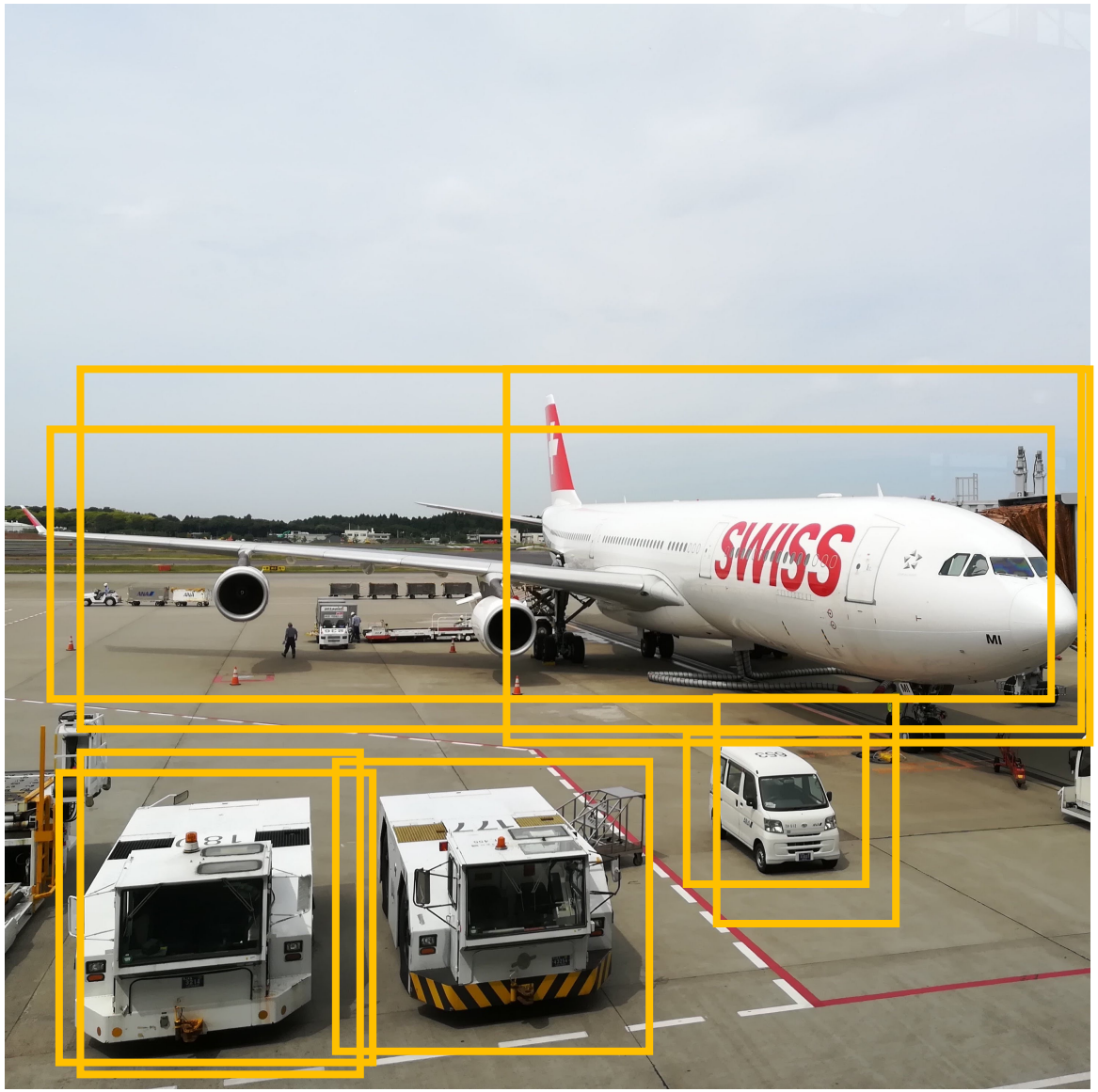

https://arxiv.org/abs/1811.04533

# Non-maximum Suppression (NMS)



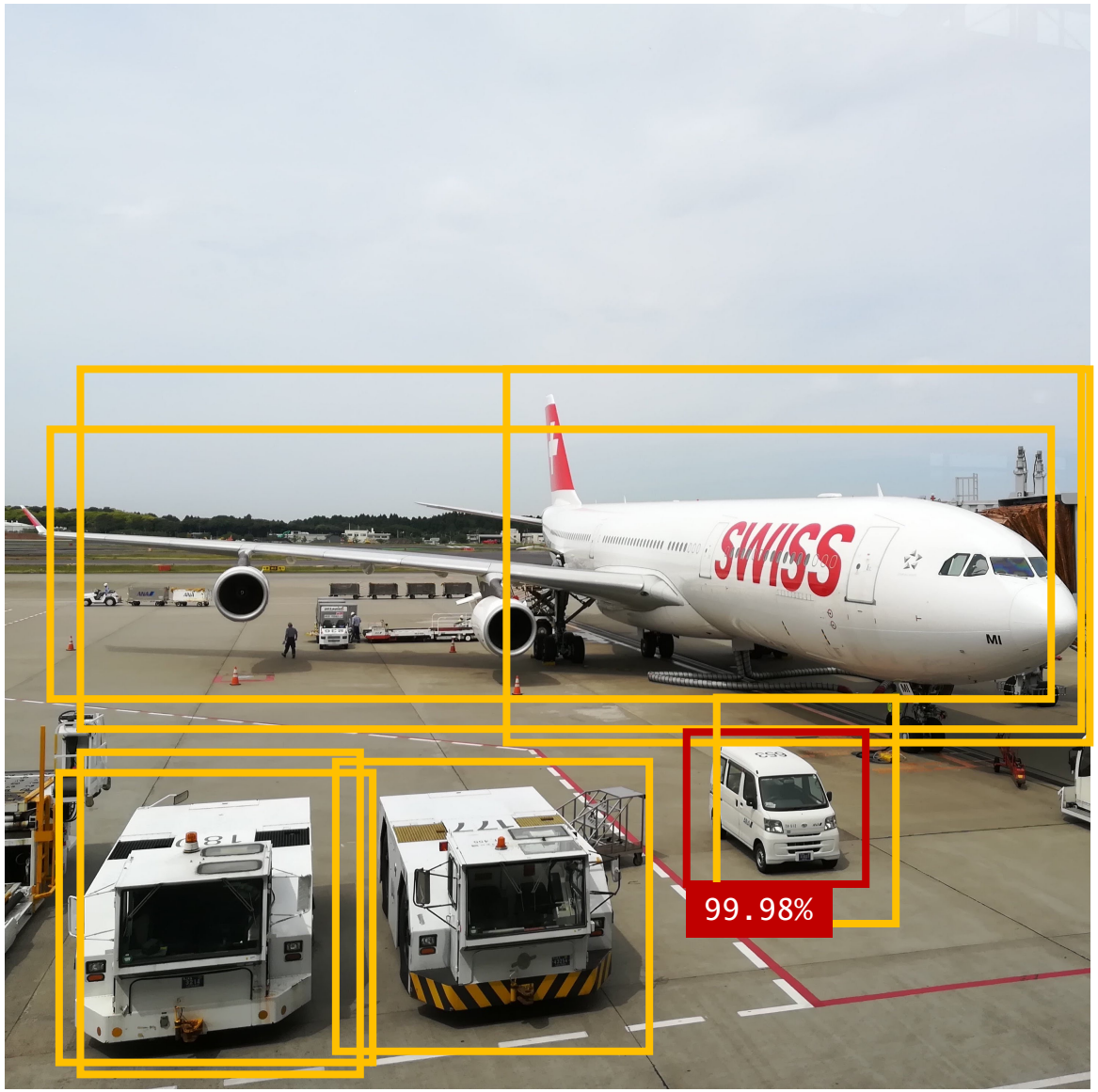

1. 予測において数多くの bbox が出力される。

# Non-maximum Suppression (NMS)



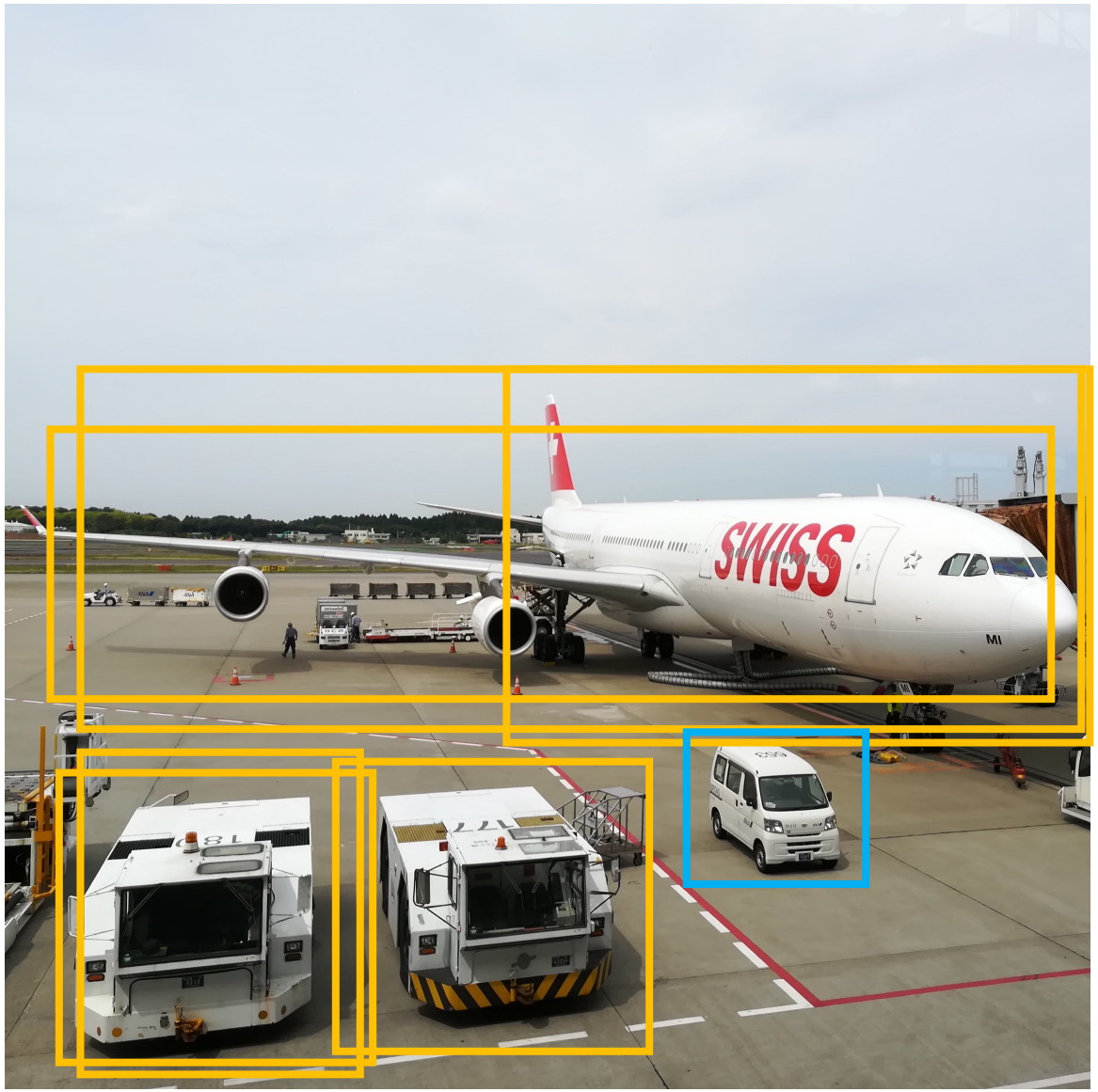

1. 予測において数多くの bbox が出力される。
2. 信頼度が最も高い k 個の bbox を残す。

# Non-maximum Suppression (NMS)



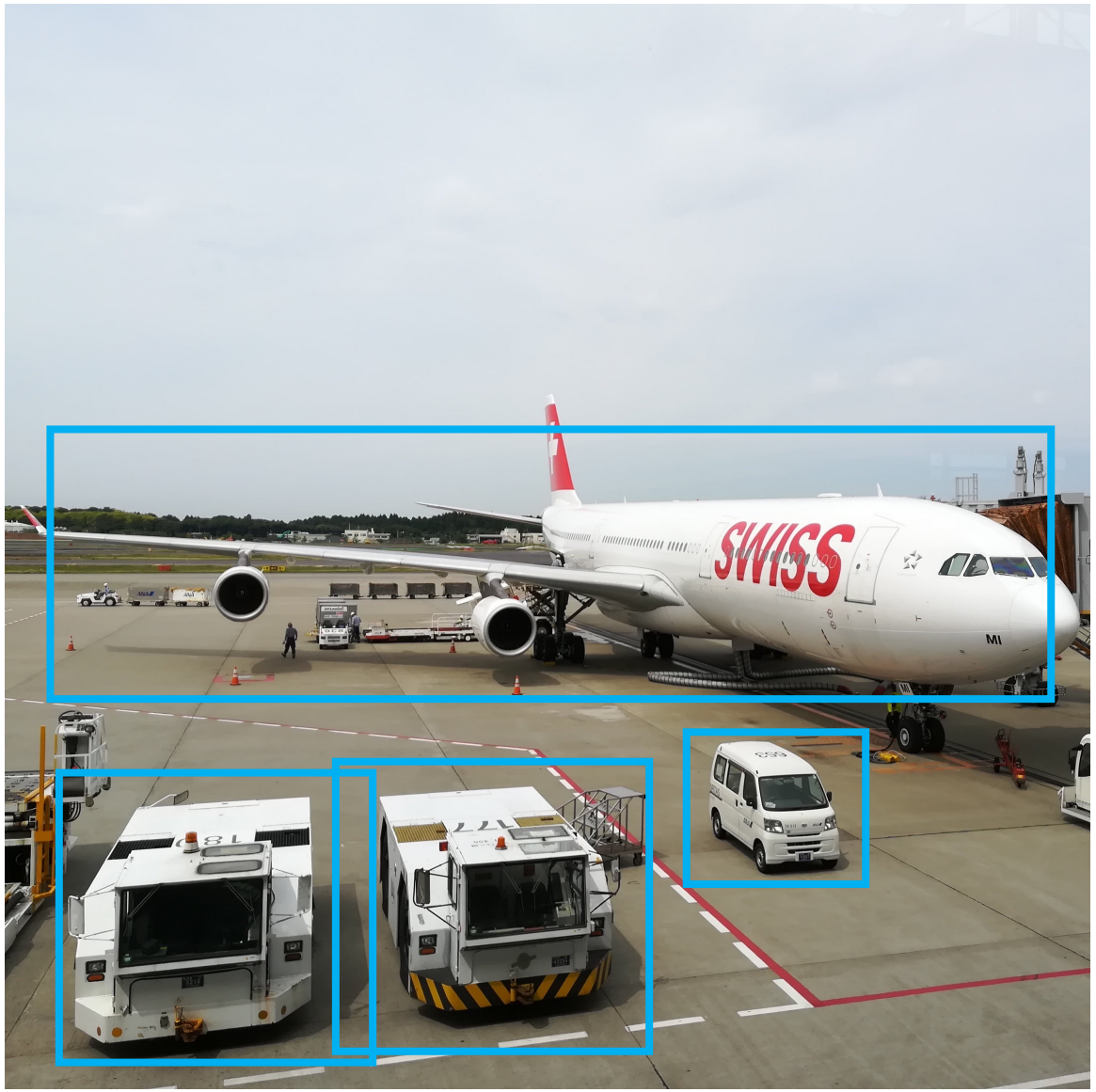

1. 予測において数多くの bbox が出力される。
2. 信頼度が最も高い k 個の bbox を残す。
3. 信頼度が最も高い bbox を基準として、他の bbox との IoU を計算する。

# Non-maximum Suppression (NMS)




1. 予測において数多くの bbox が出力される。
2. 信頼度が最も高い k 個の bbox を残す。
3. 信頼度が最も高い bbox を基準として、他の bbox との IoU を計算する。
4. IoU > 0.5 ならばその bbox を除去し、基準となった bbox を予測結果として出力する。

# Non-maximum Suppression (NMS)




1. 予測において数多くの bbox が出力される。
2. 信頼度が最も高い k 個の bbox を残す。
3. 信頼度が最も高い bbox を基準として、他の bbox との IoU を計算する。
4. IoU > 0.5 ならばその bbox を除去し、基準となった bbox を予測結果として出力する。
5. 残った bbox に対して、手順 3-4 を繰り返す。

# 物体検出入門

物体分類 – torch

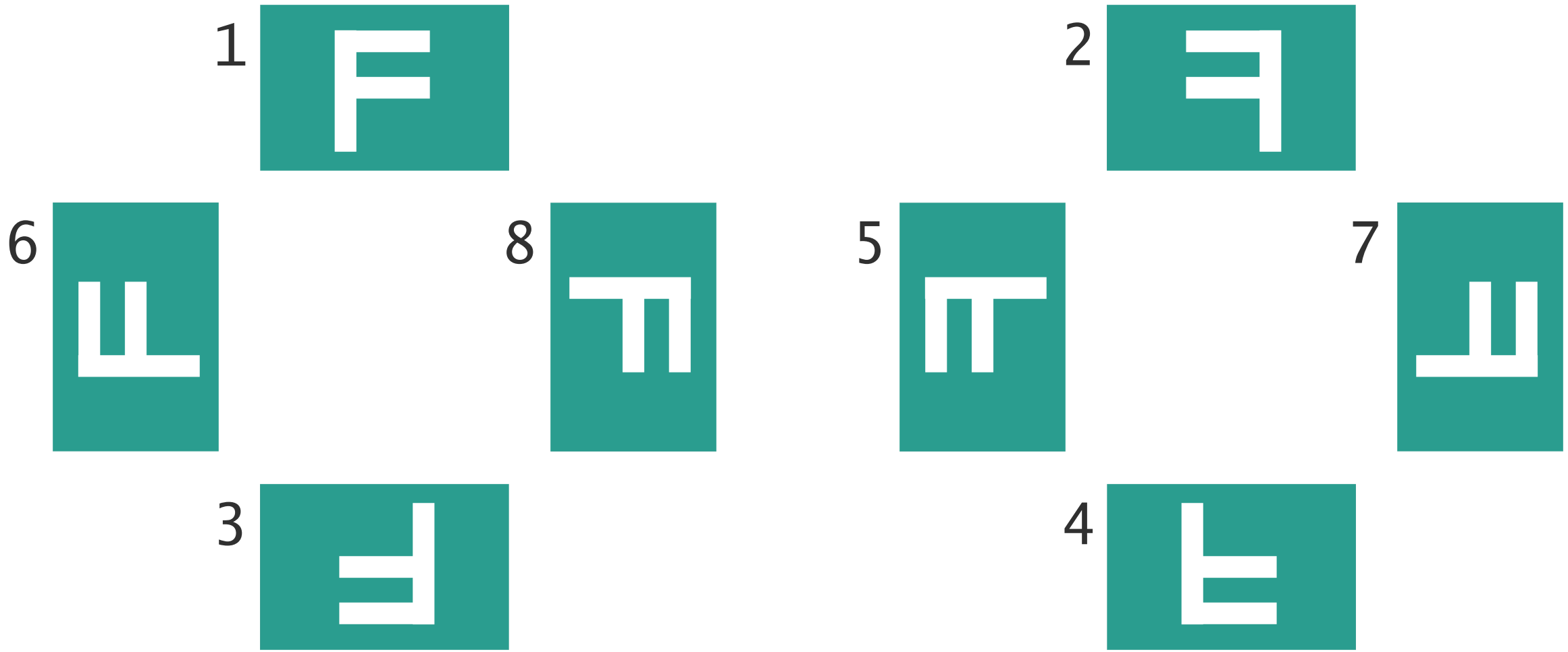
物体検出アルゴリズム

**アノテーションツール**

物体検出 – Detectron2

物体検出 – MMDetection

# EXIF Orientation



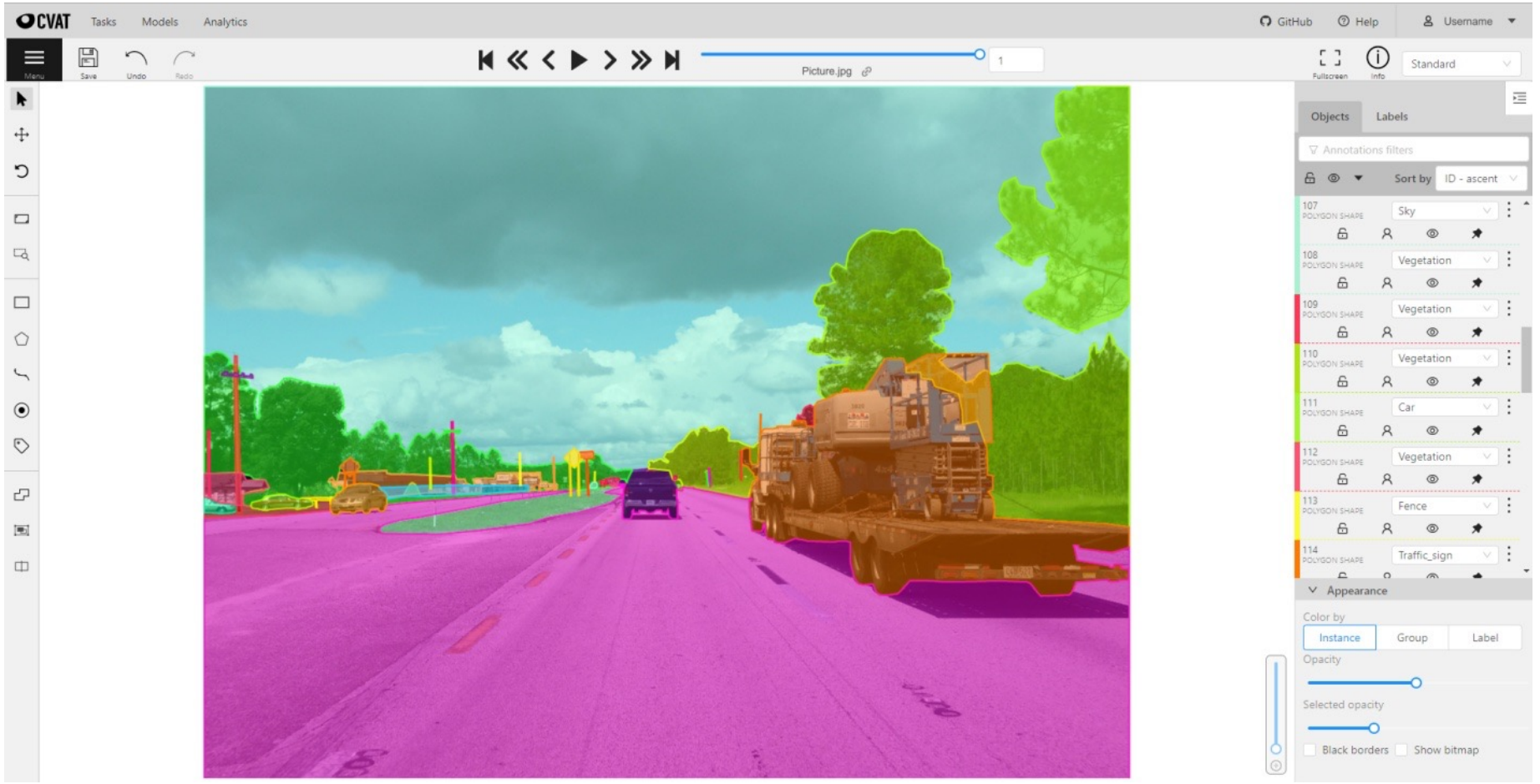
# EXIF Orientation

```
im = PIL.Image.open(img_fpath)
im.show()
```

```
im = PIL.Image.open(img_fpath)
im = PIL.ImageOps.exif_transpose(im)
im.show()
```

# アノテーションツール

- **CVAT**
- COCO Annocator
- VoTT
- LabelImage



# 物体検出入門

物体分類 – torch

物体検出アルゴリズム

アノテーションツール

**物体検出 – Detectron2**

物体検出 – MMDetection

# 環境構築

```
$ module load cuda/10.2/10.2.89 cudnn/7.6.5

$ mkdir ws
$ mkdir ws/tmp

# go to work directory
$ cd ws

# conda environment
$ conda create -n wsenv python=3.7
$ conda activate wsenv


# package installation
$ conda install -c conda-forge jupyterlab
$
$ conda install -c torch pytorch torchvision cudatoolkit=10.2
$
$ git clone https://github.com/facebookresearch/detectron2.git
$ python -m pip install -e detectron2
$
$ pip install --upgrade pip
$ pip install openmim
$ mim install mmdet
$
$ pip install agrolens
```

途中結果などを保存するための一時フォルダ。

Python 3.6 以上

pytroch (≥ 1.8) 関連のパッケージをインストール。

Detectron2 パッケージをインストール。

MMDetection パッケージをインストール。

お試しパッケージ。

# PyTorch

https://pytorch.org/

| | | | | |
|---|---|---|---|---|
| PyTorch Build | **Stable (1.10)** | Preview (Nightly) | LTS (1.8.2) | |
| Your OS | **Linux** | Mac | Windows | |
| Package | **Conda** | Pip | LibTorch | Source |
| Language | **Python** | C++ / Java | | |
| Compute Platform | **CUDA 10.2** | CUDA 11.3 | ROCm 4.2 (beta) | CPU |
| Run this Command: | `conda install pytorch torchvision torchaudio cudatoolkit=10.2 -c pytorch` | | | |

gsc01-07: CUDA 10.2

gsc08-10: CUDA 11.4

# Detectron2

https://github.com/facebookresearch/detectron2

- 物体検出、インスタンスセグメンテーション、キーポイント検出
- 訓練速度が速い
- インストール要件
  - Linux or macOS with Python ≥ 3.6
  - PyTorch ≥ 1.8, torchvision


Linux

```
git clone https://github.com/facebookresearch/detectron2.git
python -m pip install -e detectron2
```

macOS

```
git clone https://github.com/facebookresearch/detectron2.git
CC=clang CXX=clang++ ARCHFLAGS="-arch x86_64" python -m pip install -e detectron2
```

# MMDetection

- 物体検出、インスタンスセグメンテーション
- 対応アーキテクチャが多い
- 訓練速度が Detectron2 に比べて遅い
- インストール要件
  - Linux or macOS with Python ≥ 3.6
  - PyTorch ≥ 1.3
  - CUDA ≥ 9.2

```
pip install --upgrade pip
pip install openmim
mim install mmdet
```

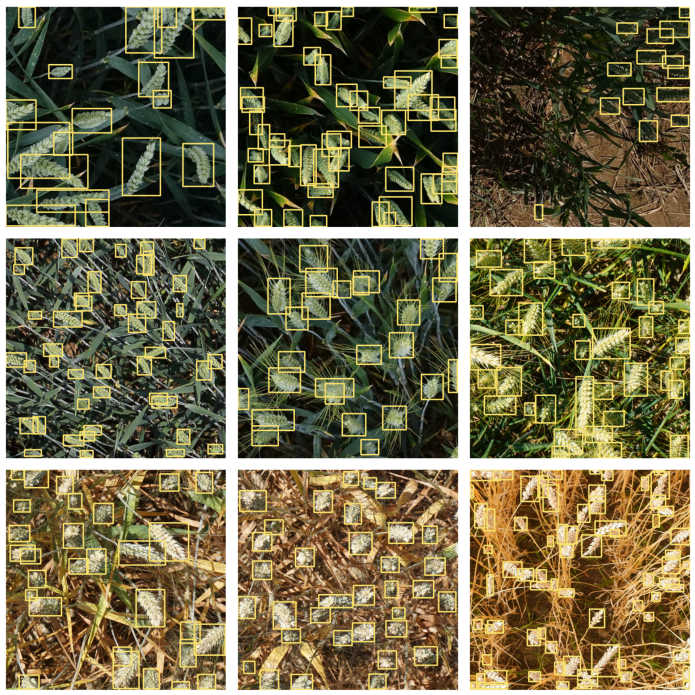
# AgroLens

- Detectron2 および MMDetection をより使いやすくした Python ライブラリ
  - 物体検出および注目物体検出
  - Faster R-CNN, YOLO3, SSD, RetinaNet, U$^2$-Net に対応
- CUI と GUI の両方で利用が可能

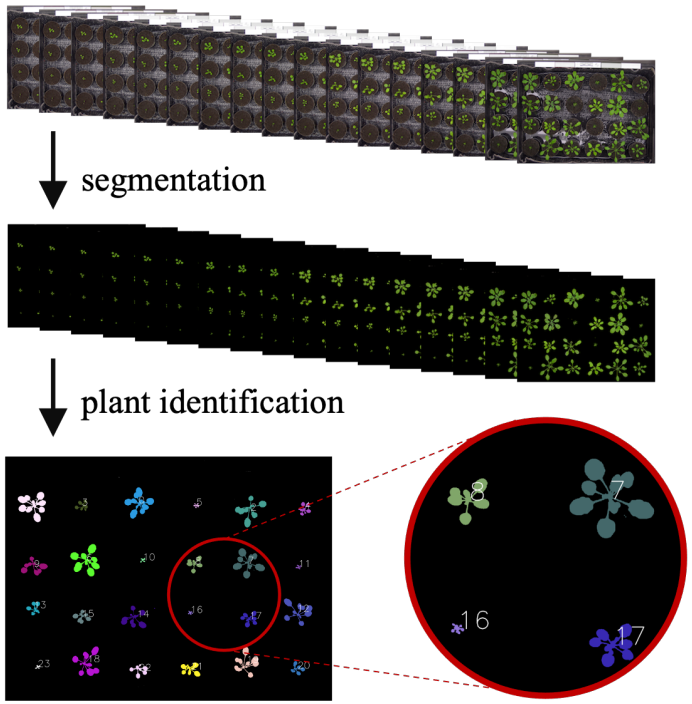
**インストール方法**

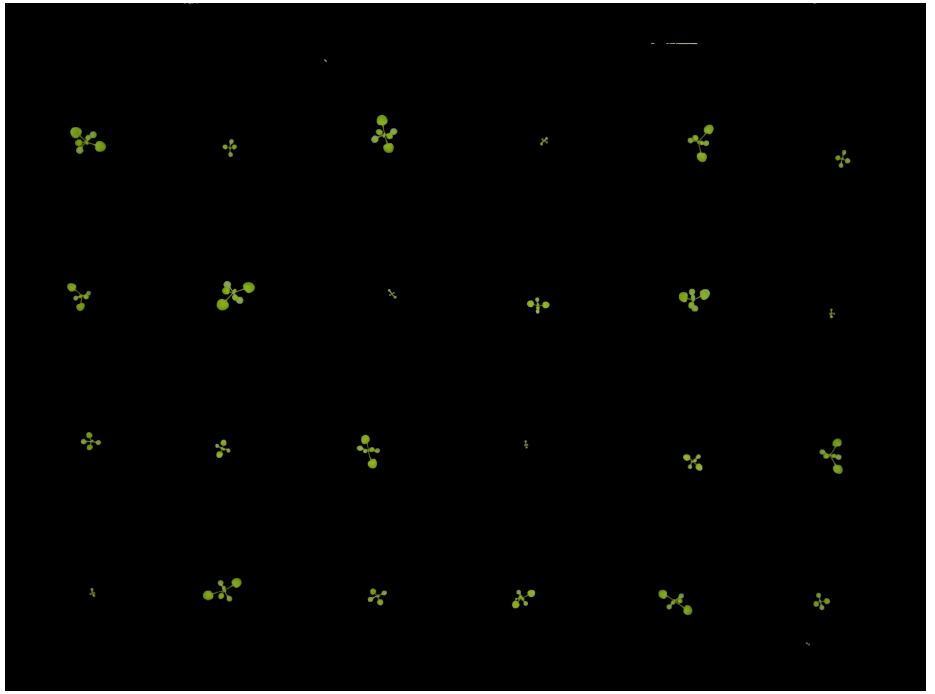
```
# torch torchvision
# detectron2
# mmdet

pip install agrolens
```

Faster R-CNN



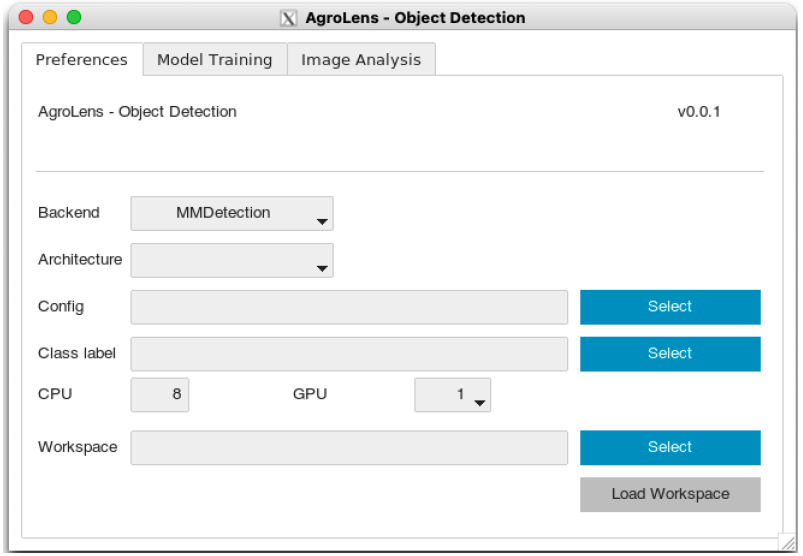
U$^2$-Net



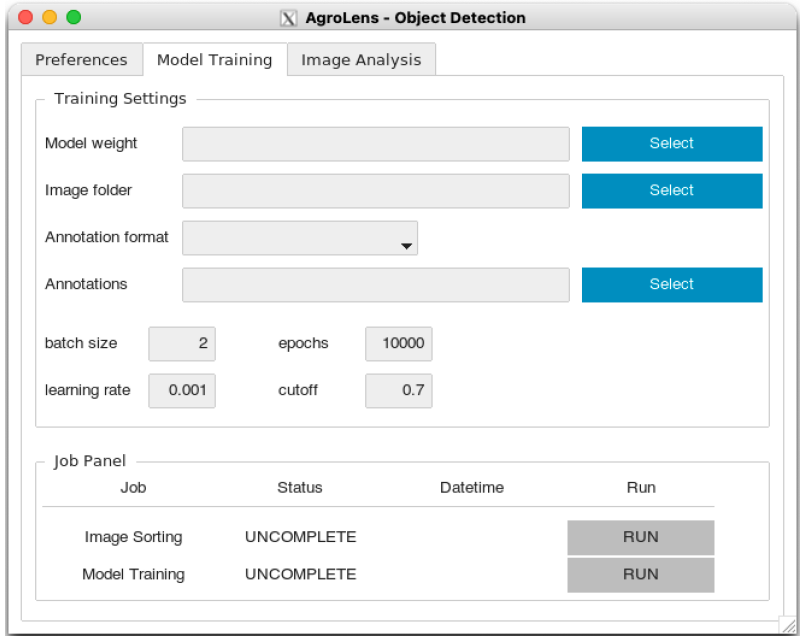



# AgroLens

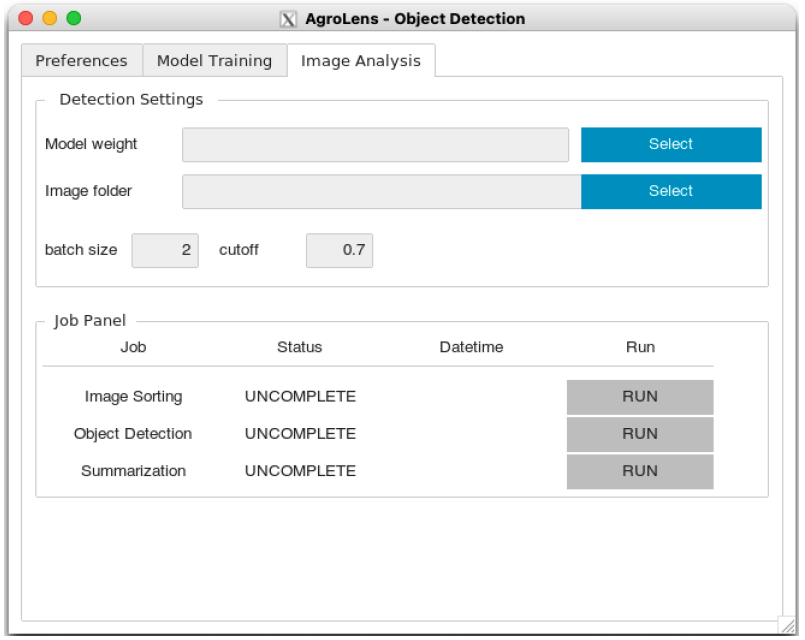
## 設定パネル




## モデル訓練パネル




## 推論パネル




## GUI アプリ起動方法

```
# object detection
agrolens od

# salient object detection
agrolens sod
```

# AgroLens API – 物体検出 モデル訓練

```
import os
from agrolens.models import FasterRCNN
from agrolens.utils import ImageAnnotation

ws = 'tmp'
images     = '/data/workshop1/odws/COCO/images'
annotation = '/data/workshop1/odws/COCO/annotations.json'
cl         = '/data/workshop1/odws/class_labels.txt'




net = FasterRCNN(cl, workspace=ws, backend='detectron2')


net.train(annotation, images,
          batchsize=8, epoch=1000,
          gpu=1, cpu=8)


net.save('trained_fasterrcnn.pth')
```

```
CUDA_VISIBLE_DEVICES=0 python train.py
```

途中結果などを保存するための一時フォルダ。

訓練用画像を含むフォルダ。

COCO フォーマットのアノテーションファイル。

1 行 1 ラベルからなるテキストファイル。

```
leaf
tree
flower
```

出力クラス数を入力して Detectron2 で Faster R-CNN アーキテクトを構築する。

画像およびアノテーションを入力して、モデルの訓練を行う。

# AgroLens API – 物体検出 推論

```
import os
from agrolens.models import FasterRCNN
from agrolens.utils import ImageAnnotation

ws     = 'tmp'
images = '/data/workshop1/odws/COCO/images'
cl     = '/data/workshop1/odws/class_labels.txt'

net = FasterRCNN(cl, model_weight='trained_fasterrcnn.pth',
                 workspace=ws, backend='detectron2')


outputs = net.inference(images, batchsize=8, gpu=1, cpu=8)


for output in outputs:
  output.draw(
      'bbox',
      os.path.join(ws, os.path.basename(output.image_path)),
      label=True, score=True
  )

outputs.format('coco', 'inference_results.json')
```

```
CUDA_VISIBLE_DEVICES=0 python infer.py
```

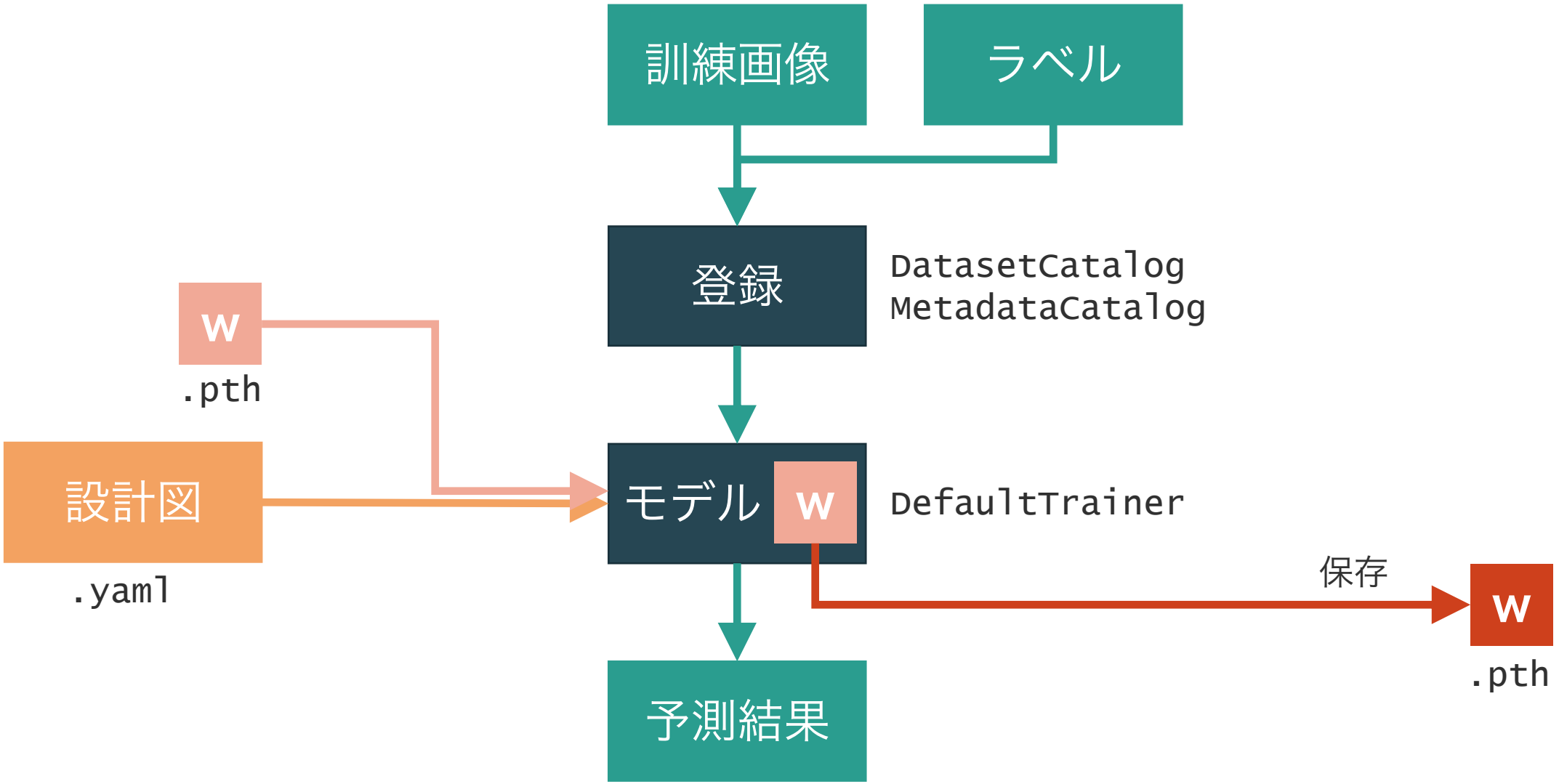
出力クラス数と訓練済みの重みを入力して Detectron2 で Faster R-CNN アーキテクトを構築する。

画像ファイルまたは画像が含まれているフォルダを与えて推論を行う。

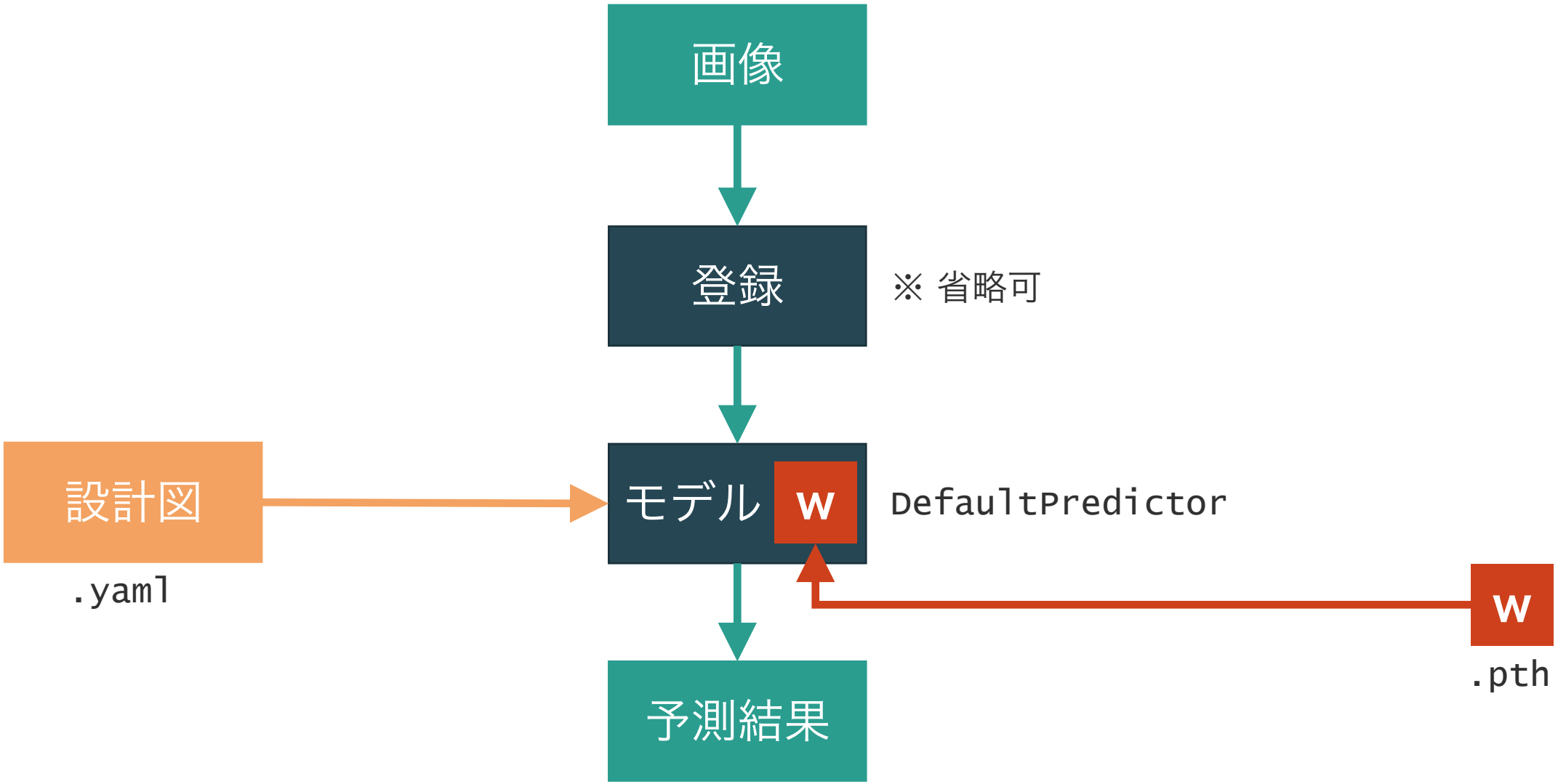
推論結果を画像として出力し、検出されたオブジェクトにバウンディングボックスを付け、予測されたクラスおよびスコアも表示する。

推論結果を COOO フォーマットで保存する。

# Detectron2 – 訓練プロセス



# Detectron2 – 推論プロセス



# 練習問題

Global Wheat Head Dataset を利用して、小麦の穂検出モデルを作成してみよう。

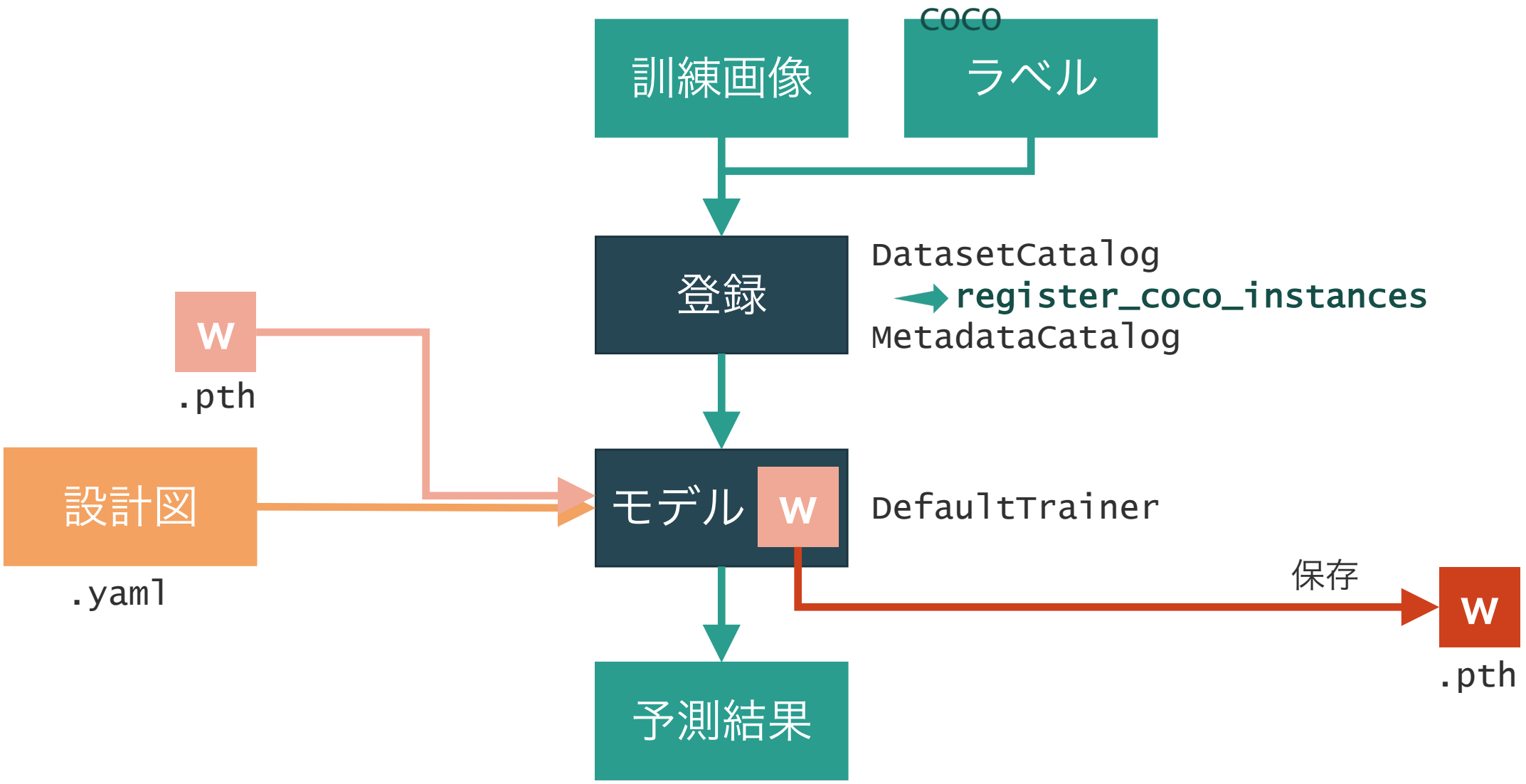
```
$ ls /data/workshop1/rcait/dataset/gwhd
train train.csv

$ ls /data/workshop1/rcait/dataset/gwhd/train | head
000be778dbb878bf22db3a3525d9722d7f2a49a3a1fb49321c1a43beabd88c06.png
001be48ccfab560e650dc0a4cceeb1636844520c88267e215ec9987688566931.png
003c89679439f746a046c4a89aad17ff32d4e0fe0be96c17518a8f972f31b30d.png
007b9fa7360dd0e511b7663287b3237299123b03380da0ee90de998b7d5c573f.png
 :
 :

$ head /data/workshop1/rcait/dataset/gwhd/train.csv
image_name,BoxesString,domain
7b73239dfd89b06c03e1be81cc5074ec47ae048305ec6377b692a7df579723d1,949 967 999 994;368 649 443 685;118 280 185
333;899 564 947 609;604 243 682 297;178 613 220 647;18 598 71 655;96 929 119 957;503 588 558 710;800 221 853
280;344 792 398 844;323 115 355 157;161 924 278 1018;587 303 645 358;48 970 120 1022;657 696 723 742,0
0e37ccf64ccbd456f07dcfe110133018f324e5bdf63b086b35c262079f142426,691 60 730 119;838 901 897 969;477 444 526
489;406 837 441 877;661 471 695 514;456 400 504 437;171 175 198 229;988 461 1023 487;520 53 560 99;938 946 990
1015;362 5 412 32;588 512 650 553;941 452 974 473;649 339 693 396;427 374 483 424;756 576 789 626;485 306 532
360;462 807 508 846; 89 901 117 937;268 52 293 75;719 210 764 277;901 569 972 620;799 475 878 535;0 349 32
375;816 379 859 419;276 742 320 776,0
89e0aa4148f0a9ff01e9e55c5a2bcbe294150444dbe43c303e399c02ed6d9142,953 178 1023 24456 196 129 250;612 557 666
616;752 22 807 66;594 872 698 990;355 278 400 330;303 702 347 777;390 263 443 310;827 646 884 703;938 758 101
 :
 :
```

# 解答

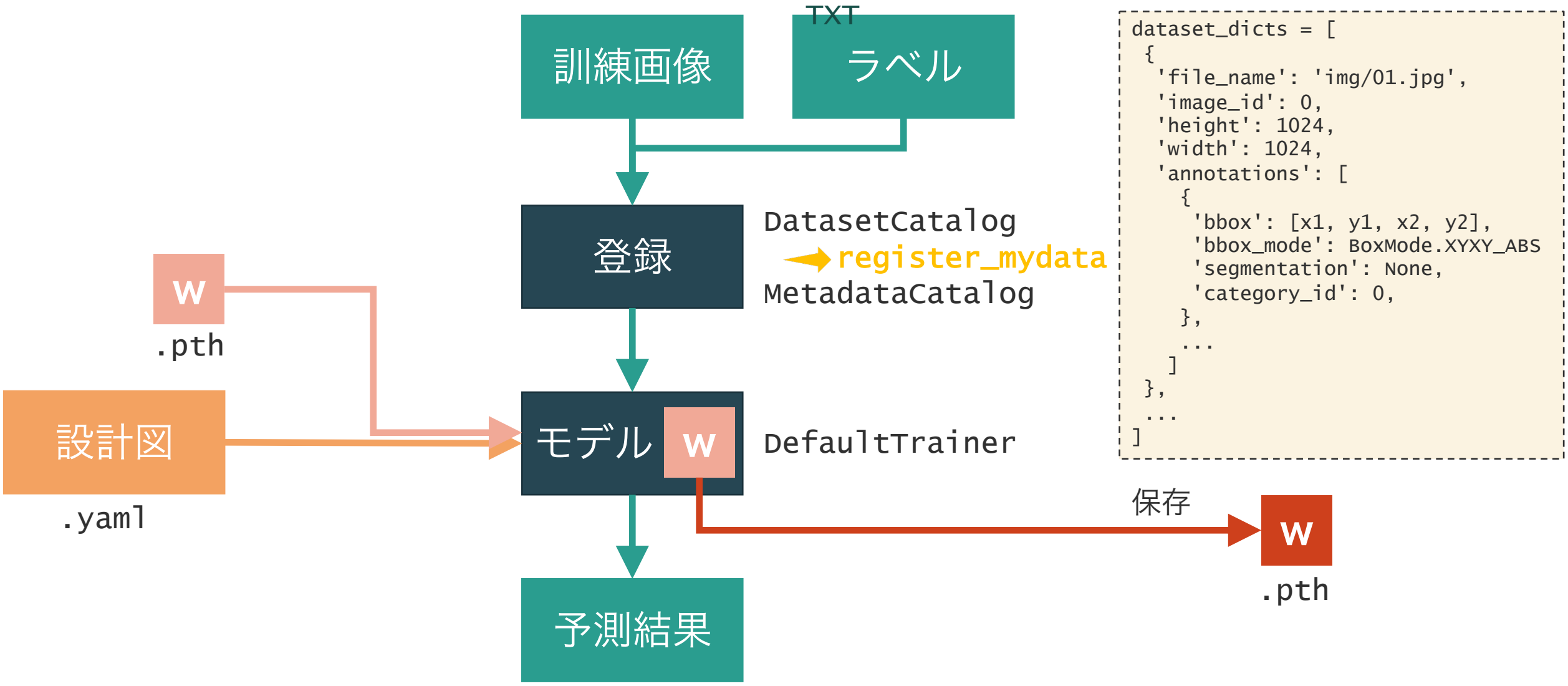
Global Wheat Head Dataset を利用して、小麦の穂検出モデルを作成してみよう。

```
$ cd
$
$ cp -r /data/workshop1/odws/gwhd_ans gwhd
$ cd gwhd
$
$ ln -s /data/workshop1/rcait/dataset/gwhd/train .
$ ln -s /data/workshop1/rcait/dataset/gwhd/train.csv .
$
$ python gwhd2coco.py ./train ./train.csv ./train.json
$
$
$
$ # run script directly
$ CUDA_VISIBLE_DEVICES=0 python train_fasterrcnn.py
$ CUDA_VISIBLE_DEVICES=0 python predict_fasterrcnn.py
$
$
$ # run as qsub job
$ bash run_script.sh
$
$
$
$
$
$
$
```

# Detectron2 – 訓練プロセス – COCO フォーマット



# Detectron2 – 訓練プロセス – 独自フォーマット

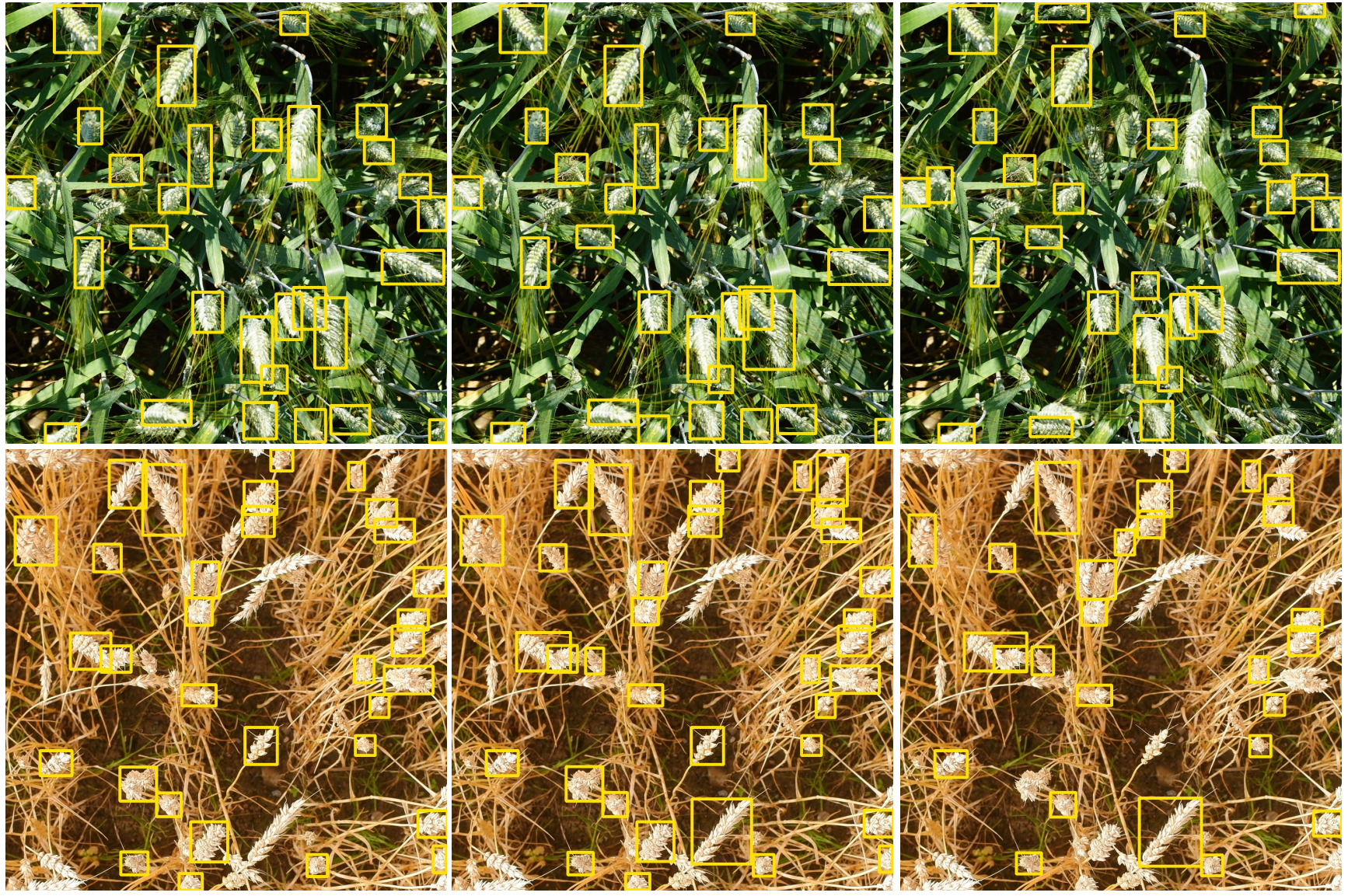


# Detectron2 – GWHD 検出結果

faster R-CNN
(COCO format)

faster R-CNN
(TXT format)

RetinaNet
(COCO format)



# 物体検出入門

物体分類 – torch

物体検出アルゴリズム

アノテーションツール

物体検出 – Detectron2

**物体検出 – MMDetection**