薬の良し悪しを見分ける

製薬業界におけるコンピューター・ビジョン

Eduardo Alvarez インテル コーポレーション シニア AI ソリューション・エンジニア

このチュートリアルでは、AI リファレンス・キットの Visual Quality Inspection（英語）を使用して、コンピューター・ビジョン・ソリューションを構築する方法を紹介します。このリファレンス・キットは、製造工程における不良品の視覚的検査の概念を説明するさまざまなデータセット（英語）へのリンクを提供します。製造工程で発生するさまざまな不良品で試すことができますが、このチュートリアルでは錠剤の品質に注目します。このデータセットでは、市販のサプリメントが「良」または「不良」のカテゴリーに分類されています。ここでは、このデータセットを使用して、事前にトレーニングした VGG-16 モデルを転移学習し、錠剤の自動品質管理ツールを作成します。

VGG-16 は、16 層の深さを持つ畳み込みニューラル・ネットワークです。ILSVRC 2014 で最も成績の良いアーキテクチャーの 1 つでした。クラス分類タスクでは、(分類エラー率 6.66% の GoogleNet に次ぐ) 2 位となり、トップ 5 分類エラー率は 7.32% でした。また、位置特定タスクでは優勝し、位置特定エラー率は 25.32% でした。

しかし、ゼロからトレーニングすると非常に時間がかかります。VGG-16 でトレーニングした ImageNet の重みのサイズは 528MB あるため、かなりのディスク容量と帯域幅を必要とし、非効率的です。1 億 3800 万個のパラメーターは、勾配の爆発的な増加につながります。そこで、AWS* EC2* m6i.4xlarge インスタンス（第 3 世代インテル® Xeon® Platinum 8375C プロセッサー 2.90GHz）上で稼働する PyTorch* 向けインテル® エクステンション（IPEX）を利用して、錠剤データセット上で事前にトレーニングした VGGNet 分類アーキテクチャーを転移学習させます。

IPEX には、インテル® ハードウェア上で PyTorch* のパフォーマンスを向上する最適化が含まれています（**図 1**）。また、Python* の API も含まれており、ユーザーは 2、3 行のコードを変更するだけで、これらの最適化を利用できます。IPEX の最適化のほとんどは、最終的に stock PyTorch* のリリースに含まれる予定です。

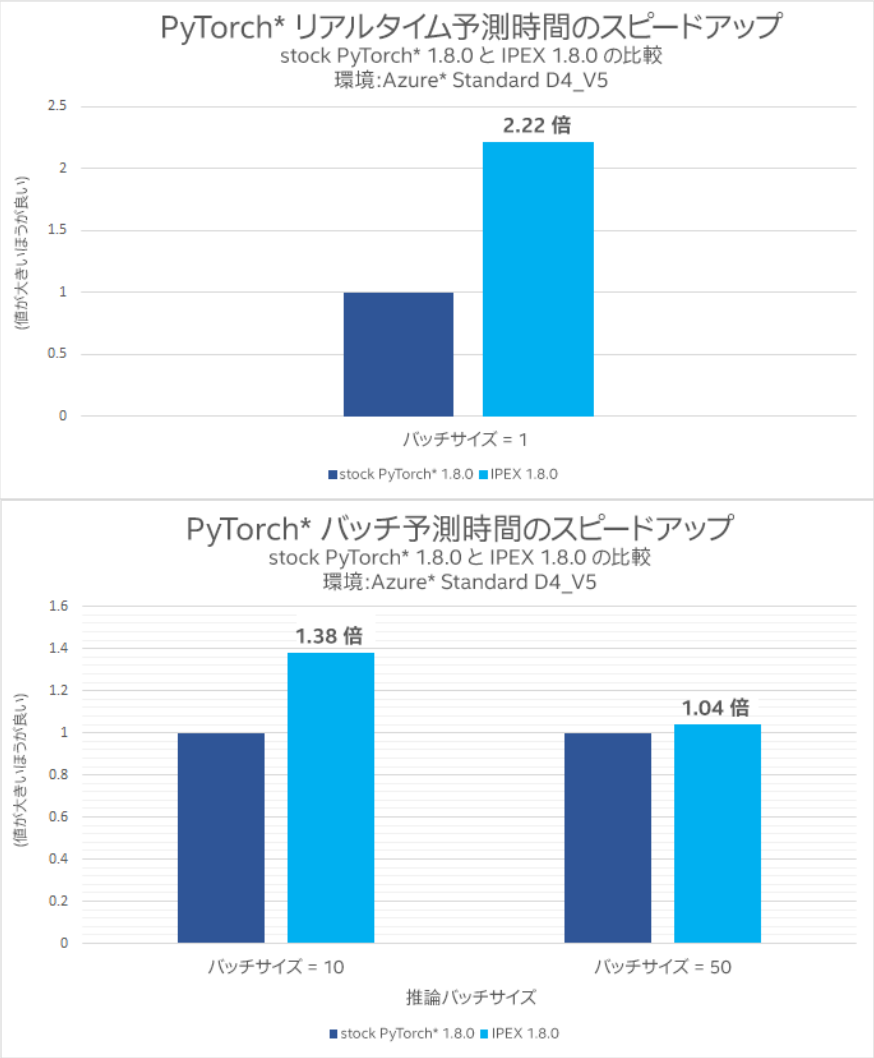

図 1. PyTorch* 向けインテル® エクステンション（IPEX）により、stock PyTorch* と比較して予測時間が向上。この比較は v1.8.0 で行っていますが、掲載時の最新バージョンは v1.12.0 です。

第 3 世代のインテル® Xeon® プロセッサーは、インテル® アドバンスト・ベクトル・エクステンション（インテル® AVX-512）で低精度の BFloat16 をネイティブにサポートし、将来的にはインテル® アドバンスト・マトリクス・エクステンション (インテル® AMX) で混合精度をサポートする予定です。インテル® AMX を使用すれば、単精度でネットワーク精度を維持したまま、半精度でトレーニングできます。

IPEX は、Conv2D + ReLU、Linear + ReLU など、頻繁に使用される演算子パターンの融合を透過的にサポートし、TorchScript でさらにパフォーマンスを最適化できます。また、演算子を最適化し、いくつかのカスタマイズされた演算子を実装します。一部の ATen 演算子は、ATen の登録メカニズムにより IPEX で最適化された演算子に置き換えられます。さらに、一般的なトポロジー向けにいくつかのカスタマイズされた演算子が実装されています。例えば、ROIAlign と NMS は Mask R-CNN で定義されています。IPEX はカスタマイズされた演算子も最適化し、これらのトポロジーのパフォーマンスを向上します。

探索的データ分析

製薬業界において品質管理は非常に重要です。良 (**図 2**) と不良 (**図 3**) の錠剤のさまざまな側面を見てみましょう。不良品の例としては、色、斑点 / 汚れ、割れ / 欠け、刻印の不具合、錠剤の種類違い、傷などが挙げられます。

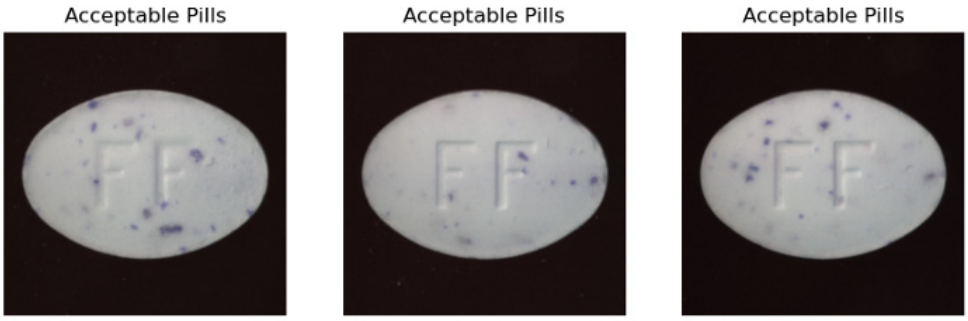

図 2. 良 / 許容範囲の錠剤の例

図 3. 不良 / 欠陥錠剤の例

平均的な良と不良の錠剤の差分分析では、錠剤の右上に欠陥があり、錠剤の右側に変色のようなものがあることがわかります（**図 4**）。これは、一部の錠剤の右側面にダメージを与える製造工程の問題を示している可能性があります。

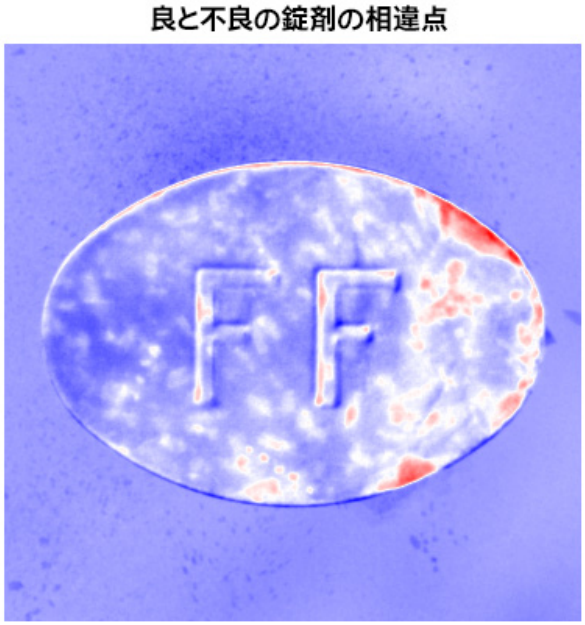

図 4. 平均的な許容範囲の錠剤と平均的な不良の錠剤の相違点

カスタム VGG-16 モデル定義

ImageNet で事前学習された VGG-16 特徴抽出器と、カスタム分類ヘッドを使用して、カスタム・マルチクラス分類モデルを定義します。転移学習を可能にするため、最初の畳み込みブロックのパラメーターはフリーズ（固定）されます。この関数は、クラススコア（トレーニング・モードの場合）、クラス確率、そして正規化された特徴マップ（評価モードの場合）を返します。

PyTorch* モデルのトレーニング・スキーマを初めて目にした方は、いくつかの定型コードを記述する必要があることに気付くでしょう。PyTorch* のモデルは通常、以下のコンポーネントを必要とします。

- **データの前処理:**データの読み込みとカスタマイズのため、PyTorch* の「Dataset クラス」を拡張します。これは、フォルダーから画像、それぞれのラベル、その他必要なメタデータを抽出するのに便利です。
- **データの読み込み:**PyTorch* は「DataLoader クラス」も提供しており、モデルのトレーニングと評価時に「Dataset クラス」の操作を支援します。この関数は、トレーニングと推論中にモデルにデータを提供する役割を担うジェネレーターです。
- **モデルの定義：**PyTorch* モデルを定義するには、「Module クラス」を拡張するクラスを定義します。コンストラクター（`__init__`）はモデルの層を決定し、`forward()` 関数は定義されたモデルの層を通じて入力データを前方伝播させる方法を定義する責任があります。
- **トレーニング関数：**トレーニング・コンポーネントの損失関数と最適化アルゴリズムを定義する必要があります。

以下のコード例は、この例のモデル定義を表しています（**図 5**）。コンストラクターでは、拡張する VGG-16 モデルと、既存のアーキテクチャーに追加するカスタム層を定義しています。 `_freeze_params` メソッドは `requires_grad` を `False` に設定することで、VGG-16 モデルの浅い層をフリーズ（転移学習のため層の重みを変更できないように）します。`forward()` メソッドは、重みを適用し、モデルを通じてデータを移動します。

```
class CustomVGG(nn.Module):

    def __init__(self, n_classes=2):
        super().__init__()
        self.feature_extractor = models.vgg16(pretrained=True).features[:-1]
        self.classification_head = nn.Sequential(
            nn.MaxPool2d(kernel_size=2, stride=2),
            nn.AvgPool2d(
              kernel_size=(INPUT_IMG_SIZE[0] // 2 ** 5, INPUT_IMG_SIZE[1] // 2 ** 5)
            ),
            nn.Flatten(),
            nn.Linear(
                in_features=self.feature_extractor[-2].out_channels,
                out_features=n_classes,
            ),
        )
        # self._freeze_params()

    def _freeze_params(self):
        for param in self.feature_extractor[:23].parameters():
            param.requires_grad = False

    def forward(self, x_in):
        """
        forward
        """
        feature_maps = self.feature_extractor(x_in)
        scores = self.classification_head(feature_maps)

        if self.training:
            return scores

        probs = nn.functional.softmax(scores, dim=-1)

        weights = self.classification_head[3].weight
        weights = (
            weights.unsqueeze(-1)
            .unsqueeze(-1)
            .unsqueeze(0)
            .repeat(
                (
                    x_in.size(0),
                    1,
                    1,
                    INPUT_IMG_SIZE[0] // 2 ** 4,
                    INPUT_IMG_SIZE[0] // 2 ** 4,
                )
            )
        )
        feature_maps = feature_maps.unsqueeze(1).repeat((1, probs.size(1), 1, 1, 1))
        location = torch.mul(weights, feature_maps).sum(axis=2)
        location = F.interpolate(location, size=INPUT_IMG_SIZE, mode="bilinear")

        maxs, _ = location.max(dim=-1, keepdim=True)
        maxs, _ = maxs.max(dim=-2, keepdim=True)
        mins, _ = location.min(dim=-1, keepdim=True)
        mins, _ = mins.min(dim=-2, keepdim=True)
        norm_location = (location - mins) / (maxs - mins)

        return probs, norm_location
```

図 5. カスタム VGG-16 モデルクラス

PyTorch* 向けインテル® エクステンションによる転移学習

前述したように、IPEX を使用して、錠剤データセット上で事前にトレーニングした VGGNet 分類モデルを転移学習させます。以下のコード例の 11 行目は、モデルにさまざまな最適化を適用する IPEX `optimize` メソッドの呼び出しを示しています(**図 6**)。IPEX のモデルへの適用については、公式ドキュメント(英語)を参照してください。

```
# モデルのトレーニングを開始

# モデルのトレーニング
# DL アーキテクチャー、オプティマイザー、損失関数の初期化
model = CustomVGG()
class_weight = torch.tensor(class_weight).type(torch.FloatTensor).to(DEVICE)
criterion = nn.CrossEntropyLoss(weight=class_weight)
optimizer = optim.Adam(model.parameters(), lr=LR)

# IPEX 最適化
model, optimizer = ipex.optimize(model=model, optimizer=optimizer, dtype=torch.float32)

# モジュールのトレーニング
start_time = time.time()
trained_model = train(train_loader, model=model, optimizer=optimizer, criterion=criterion, epochs=EPOCHS,
    device=DEVICE, target_accuracy=TARGET_TRAINING_ACCURACY)
train_time = time.time()-start_time

# hdf5 PyTorch* モデルの保存
model_path = f"{subset_name}.h5"
torch.save(trained_model, model_path)
```

図 6. モデルのトレーニング

トレーニング済みモデルのアーキテクチャーを評価し、トレーニング済みパラメーターの総数を確認します (**図 7**)。IPEX でモデルを最適化すると、適切な層に `_IPEX` というプリフィクスが追加されます。これは、IPEX が実装されていることを確認する良い方法です。

```
from torchsummary import summary
summary(trained_model, (3, 224, 224))
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
       _IPEXConv2d-1         [-1, 64, 224, 224]           1,792
              ReLU-2         [-1, 64, 224, 224]               0
       _IPEXConv2d-3         [-1, 64, 224, 224]          36,928
              ReLU-4         [-1, 64, 224, 224]               0
         MaxPool2d-5         [-1, 64, 112, 112]               0
       _IPEXConv2d-6        [-1, 128, 112, 112]          73,856
              ReLU-7        [-1, 128, 112, 112]               0
       _IPEXConv2d-8        [-1, 128, 112, 112]         147,584
              ReLU-9        [-1, 128, 112, 112]               0
        MaxPool2d-10          [-1, 128, 56, 56]               0
      _IPEXConv2d-11          [-1, 256, 56, 56]         295,168
             ReLU-12          [-1, 256, 56, 56]               0
      _IPEXConv2d-13          [-1, 256, 56, 56]         590,080
             ReLU-14          [-1, 256, 56, 56]               0
      _IPEXConv2d-15          [-1, 256, 56, 56]         590,080
             ReLU-16          [-1, 256, 56, 56]               0
        MaxPool2d-17          [-1, 256, 28, 28]               0
      _IPEXConv2d-18          [-1, 512, 28, 28]       1,180,160
             ReLU-19          [-1, 512, 28, 28]               0
      _IPEXConv2d-20          [-1, 512, 28, 28]       2,359,808
             ReLU-21          [-1, 512, 28, 28]               0
      _IPEXConv2d-22          [-1, 512, 28, 28]       2,359,808
             ReLU-23          [-1, 512, 28, 28]               0
        MaxPool2d-24          [-1, 512, 14, 14]               0
      _IPEXConv2d-25          [-1, 512, 14, 14]       2,359,808
             ReLU-26          [-1, 512, 14, 14]               0
      _IPEXConv2d-27          [-1, 512, 14, 14]       2,359,808
             ReLU-28          [-1, 512, 14, 14]               0
      _IPEXConv2d-29          [-1, 512, 14, 14]       2,359,808
             ReLU-30          [-1, 512, 14, 14]               0
        MaxPool2d-31            [-1, 512, 7, 7]               0
        AvgPool2d-32            [-1, 512, 1, 1]               0
          Flatten-33                  [-1, 512]               0
           Linear-34                    [-1, 2]           1,026
================================================================
Total params: 14,715,714
Trainable params: 14,715,714
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.57
Forward/backward pass size (MB): 218.40
Params size (MB): 56.14
Estimated Total Size (MB): 275.11
----------------------------------------------------------------
```

図 7. PyTorch* 向けインテル® エクステンション (IPEX) モデルのアーキテクチャー

ホールドアウト・テスト・データに対する予測では、モデルは写真の 2/3 が「不良」錠剤であると判断しています (**図 8**)。境界ボックスとヒートマップは、錠剤で検出された最も顕著な欠陥を強調しています。予想どおり、「FF」刻印の問題、欠け、変色が見られます。

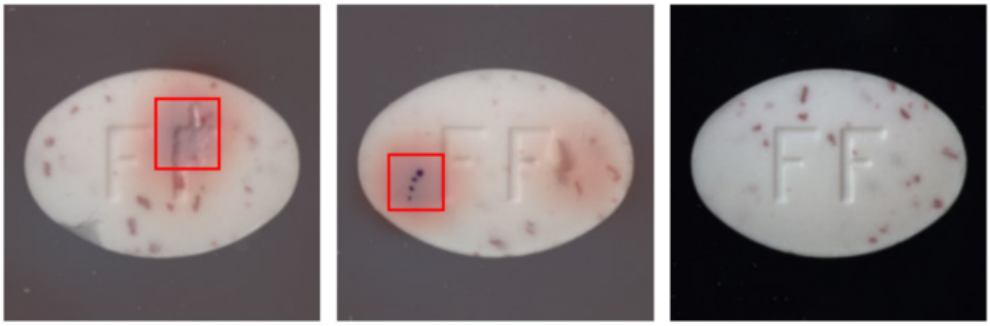

図 8. 錠剤データセットの 3 つのテスト画像に対する予測値。赤い境界ボックスで囲まれた領域は、欠陥を示しています。ヒートマップは、重要な欠陥の濃度を示しています。

まとめ

コードを少し追加するだけで、インテル® ハードウェア上でカスタム VGG-16 バイナリー分類器を最適化できました。この効率良い転移学習により、事前にトレーニングした Torch VGG-16 モデルを、錠剤データセットの画像を使用して転移学習（チューニング）し、医薬品の品質管理を行う効率良い分類ツールに変えることができました。