令和 5 年度茨城大学大学院理工学研究科情報工学専攻

修士学位論文

CLIP と姿勢推定を用いた画像内人物の動作推定

所属　情報工学専攻

著者　木村文飛 (22NM712R)

指導教員　新納浩幸教授

令和 6 年 2 月 2 日 (金)

令和 5 年度茨城大学大学院理工学研究科情報工学専攻　修士学位論文

CLIP と姿勢推定を用いた画像内人物の動作推定

著者

木村文飛 (22NM712R)

指導教員

新納浩幸教授

論文要旨

動作推定とは，センサーやカメラから得られた情報をもとに人物の動作を推定する分野である．似た分野に，姿勢推定と呼ばれる，画像内に存在する人間の骨格と顔のパーツをを推定する分野が存在する．画像内人物の動作を正確に認識することは，より高度な画像認識へと繋がるが，動作の推定をするにあたって姿勢推定だけでは不十分である．なぜなら，同じ姿勢であっても，何を持っているのか，何に座っているかで動作名は大きく変わってくるからである．周りに映る物体や場所，位置関係や前後関係といった情報を取り入れる必要がある．そのため，動作推定は画像単体から行わず，動画を用いて，位置情報の差異を計測することで物体の動きを検出し，そこから動作を特定する手法が主流である．しかし，それでは必要なデータセットが動画となってしまい，コストの増大は必至である．また，計算コストの面でも，動画を処理するとなれば，単一画像を処理する場合に比べて莫大な増加量となる．

本研究では，CLIP と姿勢推定を組み合わせた動作推定の手法を提案し，提案手法を用いることで詳細に単一画像を認識できているかを調査する．CLIP とは大量の画像とそれに対して付与されたテキストを学習した物体識別モデルである．姿勢推定には MMPOSE を用いる．CLIP を用いて画像処理を行い，周囲の状況や人と物体の相互作用を認識させ，更にそこに姿勢推定を組み合わせることで，画像内人物の動作推定の精度向上を図るというのが目的である．

実験では，提案手法と，画像処理または姿勢推定のみを使用したモデルを作成し各モデルの精度を比較した．また，精度には正解率を用いて，提案手法が動作推定に与える影響を調査した．

実験結果では，提案手法では動作推定の精度を上げることはできないことが示された．主な課題として確認できたことは，動作推定タスクに対する姿勢推定とデータセットのミスマッチが挙げられた．しかし，一部動作ではあるが，提案手法が精度を上回った例も確認されたため，本研究には未だ改善の余地があることが示された．

Master's Thesis in Scholastic 2024, Major in Computer and Information Sciences,
Granduate School of Science and Engineering, Ibaraki University

Action Estimation of Individuals in Images Using CLIP and Pose Estimation

Author : Ayato Kimura (22NM712R)

Adviser : Prof. Hiroyuki Shinnou

Abstract

Action recognition is a field that involves estimating the movements of individuals based on information obtained from sensors or cameras. In a similar domain, there is a field known as Pose Estimation, which focuses on estimating the skeleton and facial features of humans present in an image. Accurately recognizing the movements of individuals in an image leads to more advanced image recognition, but relying solely on pose estimation is insufficient for action estimation. It is essential to incorporate information such as surrounding objects, locations, and spatial relationships. Therefore, action recognition is not performed solely from individual images. By measuring differences in position information, the method identifies specific actions. However, this approach requires datasets in the form of videos, leading to inevitable increases in cost.

In this study, a methodology combining CLIP and pose estimation is proposed for action recognition. The aim is to investigate whether the proposed approach can accurately recognize details in single images. Image processing using CLIP recognizes the surrounding context and interactions between people and objects. By further integrating pose estimation, the goal is to enhance the accuracy of action recognition for individuals in images.

In experiments, the proposed methodology is compared with models using either image processing or pose estimation alone, assessing the accuracy of each model. The evaluation of accuracy employs precision, and the study investigates the impact of the proposed methodology on action recognition.

The experimental results indicate that the proposed methodology does not improve the accuracy of action recognition. The main factor is the mismatch between pose estimation and the dataset for the action recognition. However, some instances show that the proposed methodology surpasses accuracy in certain actions, suggesting there is still room for improvement in this study.

目次

第 1 章

序論

動作推定とは，センサーやカメラから得られた情報をもとに人物の行動を推定する分野である．似た分野に，姿勢推定と呼ばれる，画像内に存在する人間の骨格と顔のパーツをを推定する分野が存在する．しかし，これらの技術は似て非なる存在である．

姿勢推定はあくまで骨格を推定するだけの技術であるため，動作推定に用いることはあれども動作の推定をするにあたって姿勢推定だけでは不十分である．なぜなら，同じ姿勢であっても，何を持っているのか，何に座っているかで動作名は大きく変わってく

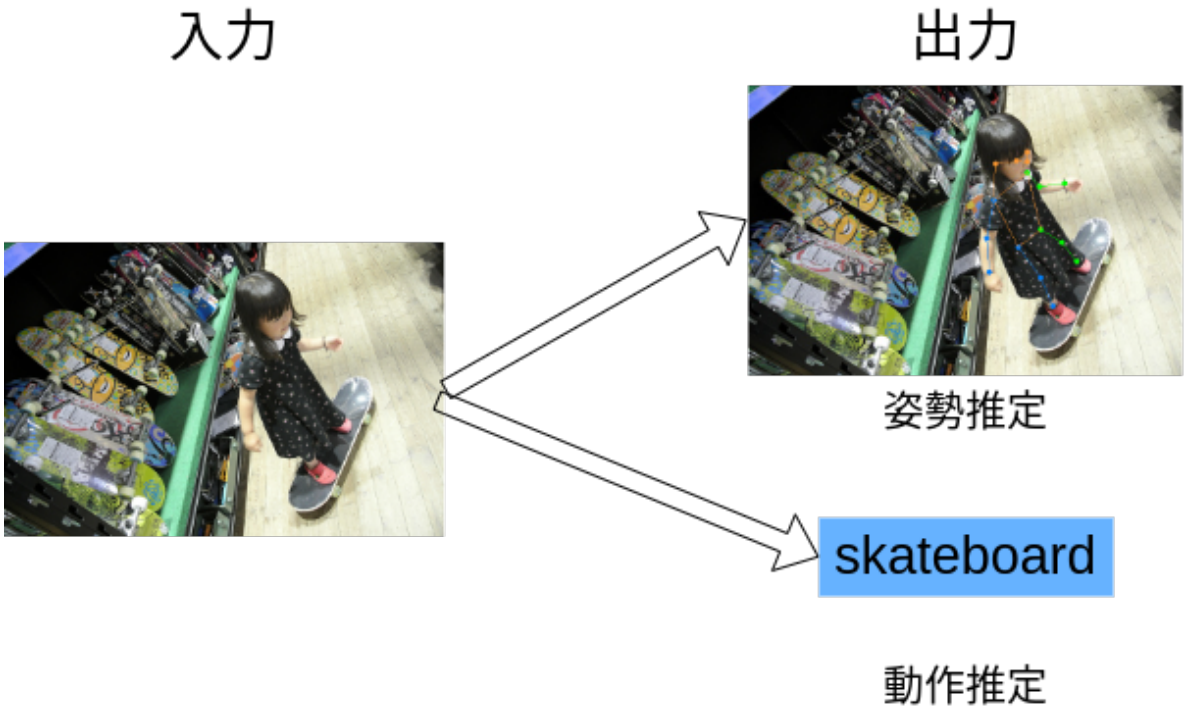

図 1.1: 姿勢推定と動作推定の違い

るからである．例えば座っている対象がバイクであればそれは“ 運転する ”という動作になり，本を持っていれば“ 読書する ”という動作になるのである．また，動作推定をするには，周りに映る物体や場所，位置関係といった情報を取り入れることも重要である．スケートボードが人間の下側に映っているなら，それは“ スケートをする ”という動作である可能性が高い，あたり一面が海なら海上である可能性が高いというように，非常に重要度の高い情報を得ることが出来る場合があるからである．

人物の動作推定の応用には，ジェスチャー，歩行，異常な姿勢，手先等様々な分野があり，これらは警備や異常検知といった監視の分野で用いられることが多い技術である．また，これらの技術は基本的に動画を用いることで，前後関係を用いた処理を行うことがほとんどである．対して，単一画像から動作推定をすることは，それらの情報を使うことが出来ないため非常に困難である．しかし，逆説的に言えば，単一画像のみを用いて正確な動作推定をすることが出来るなら，それは元のモデルより詳細に画像認識が出来ているといえるはずである．

本研究では以上のことに着目し，CLIP と姿勢推定を組み合わせた動作推定を行うことで，詳細に画像を認識できているかを調査する．CLIP とは大量の画像とそれに対して付与されたテキストを学習した物体識別モデルである．姿勢推定には MMPose を用いる．手法の概要としては，単一画像に対して CLIP を用いて画像処理を行い，周囲の状況や人と物体に関する特徴量を取得する．更にそこに姿勢推定を組み合わせることで，画像内人物の骨格情報を取得する．これら二つのベクトルを組み合わせることで，人体の姿勢情報と画像全体の情報を融合させ，動作推定を行う．ベースラインとなる，CLIP のみを用いた動作推定に対して，情報を組み合わせた動作推定の精度が高ければ，それは画像をより詳細に認識できていると考える．

第 2 章

関連研究

2.1 姿勢推定ライブラリ OpenPose を用いた機械学習による動作識別手法の検討

高崎らは，姿勢推定を用いた動作識別手法の比較を行うことで，特徴量データのみを使用して動作を識別できるかを調査した．[1]

深層学習を用いた関節点のキーポイント推定をするにあたって，OpenPose を採用している．OpenPose を用いて，1 枚の静止画による動作推定と，時系列を考慮した 10 枚の静止画を用いた動作推定を行った．

データセットには，STAIR Actions [2] を使用した．STAIR Actions は，人間の動作とそれに対応する約 5 秒の動画をまとめたデータセットである．100 種類の動作が存在し，1 つにつき 900〜1800 の動画が存在するため，動画データセットとしては非常に大規模であることが特徴である．研究で使用したのは，writing, reading newspaper, bowing という 3 カテゴリの動作のみであり，全体で約 2.7 万のデータ数である．また，動画を 1 秒ほど抜き出し，それを 10 分割することで，時系列を考慮した静止画を作成し使用した．

1 枚の静止画を用いた動作推定の手順としては，まず静止画に対して OpenPose を用いることでキーポイントを取得する．その後，キーポイントを訓練データ 7 割，テストデータ 3 割に分割し，ロジスティック回帰，ランダムフォレスト，Support Vector Machine，Neural Network の 4 手法で動作推定精度を比較した．尚，OpenPose が出力するキーポイントは，1 画像に付き 25 個であり，2 次元座標であるため，訓練データは次元数が 50 のベクトルである．時系列を考慮した 10 枚の静止画を用いた手法では，10 枚の画像から

	training	validation
(1) ロジスティック回帰	0.688	0.640
(2) ランダムフォレスト	1.000	0.786
(3) SVM	1.000	0.454
(4) NN	1.000	0.828
(4a) NN w/ Dropout	0.987	0.820
(4b) NN w/ BN	1.000	0.842
(4c) NN w/ Dropout, BN	0.970	0.813

図 2.1: 各種法による動作の識別精度 引用元 https://db-event.jpn.org/deim2019/post/papers/174.pdf

ベクトルを取得するため，それを時系列順につなげることで使用するデータを次元数 500 のベクトルとしている．論文では，1 枚の静止画を用いた動作推定の精度比較では NN の精度が 0.82 と最も高ことが示されている．このことから，動作推定のアプローチとして NN は有効であることが結論付けられている．

第 3 章

姿勢推定

3.1 概要

姿勢推定とは，動画，画像を入力として人物や動物の間接点を推定し，それを結ぶことで対象の姿勢を出力するタスクである．元々は画像から特徴点を検出する技術の総称である，キーポイント検出という分野の 1 つであり，顔に特化すれば表情分類，車に特化すれば形状分類など応用例は多岐にわたる．それを関節点に特化させたのが姿勢推定である．

姿勢推定は 2012 年に CNN が開発されたことを契機に，ディープラーニングを用いることでさらなる発展をしてきた．特に，画像内の人物が 1 人であり，全身が写っている場合は，現状のモデルで非常に高い精度で推定を行うことができる．しかし，実際に推定を行う画像には画像内に複数人が写っている場合や，人物同士の重なりがある場合が存在する．そういった画像から姿勢を推定できるようにするというのが，姿勢推定タスクにおける現状の課題である．

ディープラーニングを用いたキーポイントの出力方法に関しては大まかに 2 つ存在する．それはキーポイントの位置をそのまま出力する回帰と，ピクセル単位で確率を出力し，それが最も高い点をキーポイントとして出力するヒートマップである．

手法としては主に Top Down と Bottom Up という 2 つのアプローチが存在する．Top Down アプローチは，人を検知してからそのキーポイントを推定するアプローチである．複数人が映っている画像なら，まず人を検出し，分けてから一人ずつ姿勢を推定していく．人ごとに画像を切り取り，それを拡大できるため，Bottom Up アプローチに比べて人物の大小の影響を受けづらいというメリットがある．しかし，正常な人物の検知

を行わなければならないため，重なりや見えない部分が多い場合は精度が低下するというデメリットが存在する．

Bottom Up アプローチは，先にキーポイントを推定してから，それをグループ化する手法である．物体検出を行わないため Top Down アプローチに比べて高速であり，また人数が多い場合もそれに影響されにくいというメリットが存在する．しかし，映っている人が小さい場合は画像の加工を行わないため推定が困難というデメリットが存在する．

3.2　評価手法

姿勢推定モデルの評価に使われる代表的な評価手法として mAP（mean Average Precision）が挙げられる．mAP は物体検出でも使われる評価手法であり，選出したものの領域に対して使われるが，姿勢推定では出力したキーポイント群に対して使用する．

mAP を算出するにあたって，姿勢推定では先に OKS（Object Keypoint Similarity）と呼ばれる値を定義する．これは，出力したキーポイントと，正解となるキーポイントの距離がどれだけ近いかを 0 から 1 の数値で表す値である．以下に OKS の定義式を示す．

$$OKS = \frac{\sum_{i=0}^{N} \exp(-{d_i}^2/2s^2{k_i}^2)\delta(v_i > 0)}{\sum_{i=0}^{N} \delta(v_i > 0)}$$

N はキーポイントの総数である．d_i は i 番目のキーポイントと正解のキーポイント間の距離であり，これが小さいほど OKS の値は高くなる．s は領域の面積であり，姿勢推定の場合は人物のサイズである．k_i は関節点ごとに定められた定数である．これは人体の部位ごとにどれだけ誤差を許容するかを決める数値である．位置がブレやすい背中や臀部には，大きい値を与えることで，多少キーポイント間での誤差が生じても値が小さくなりづらくなるようにしている．v_i は関節点がアノテーションされているかを表しており，$\delta(v_i > 0)$ では関節点がアノテーションされている場合に 1 を，されていなければ 0 となるようにしている．つまり，OKS はアノテーションされているキーポイント間の距離を元にした score を平均した値である．

AP は上記の OKS が閾値を上回っているときに推定結果を正解とした場合の平均適合率である．一例として，MS-COCO データセットを用いた評価では，閾値を 0.5 から 0.95 の 10 段階に変化させ，その平均を以て mAP を算出する．

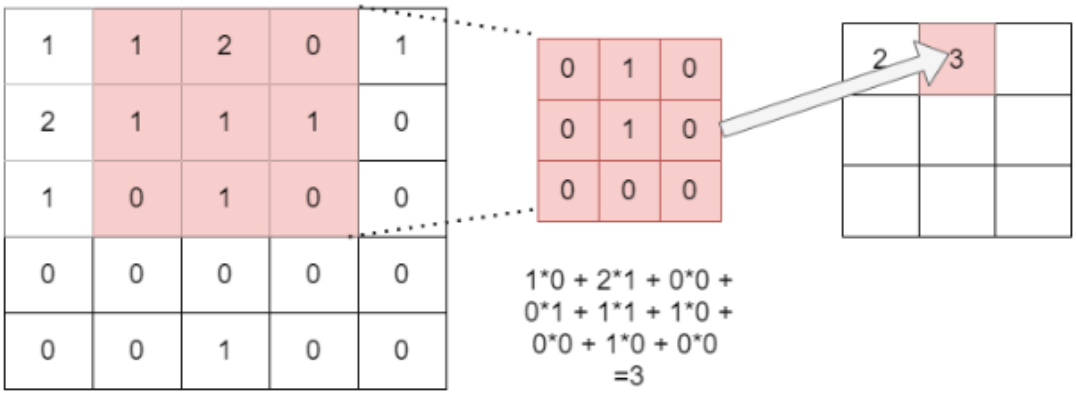

図 3.1: 畳み込み層で行われる処理の例

3.3　モデル

3.3.1　CNN

CNN（Convolutional Neural Network）とは，主に機械学習分野における画像処理において活用されるニューラルネットワークの一種であり，日本語では畳み込みニューラルネットワークと呼ばれる．CNN は画像分野において主流の手法であり，物体検出や姿勢推定，キャプション生成など活用分野は幅広い．また，自然言語処理や音響信号処理といった分野でも利用されることから，ニューラルネットワーク全体におけるオーソドックスな構造と言える．

CNN 最大の特徴は，畳み込みと呼ばれる処理を行う畳み込み層が存在する点である．畳み込み演算は，データに含まれる特徴を抽出するために行われる処理であり，空間フィルタリングの分野で用いられることが多い演算である．入力データを $I(x, y)$，カーネル（フィルタ）を $F(x, y)$ として，畳み込みを行う場合，計算式は次のように表される．

$$I'(x, y) = \sum_i \sum_j F(i, j) I(x + i, y + i)$$

具体的な計算内容としては，図 3.1 のように，対応した入力データとカーネルを掛け合わせ，その和を出力する．

CNN は，入力層，畳み込み層，プーリング層，全結合層，出力層の 5 つから構成される．入力層は，文字通り入力データを与える層である．畳み込み層では，前述した畳み込

み演算を行うことで特徴抽出をする．プーリング層では，畳み込み層の出力に対して，ある範囲内の代表値を残すダウンサンプリングを行う．様々な手法があるが，最も代表的な手法は Max Pooling であり，カーネルごとに最大値のみを残す手法である．プーリングをすることで，代表的な特徴のみを捉えながら，解像度を下げることができるため，特徴集約された出力を得ることができる．また，プーリング層にはパラメータが必要なく，かつ解像度を下げることができるため計算コストの削減につながるというメリットも存在する．全結合層では，プーリング層の出力を用いた重み付き和を計算し，それを出力層で予測結果として出力する．

CNN は，2012 年に AlexNet [3] という CNN を用いたモデルが，ILSVCR という画像処理の精度を競う大会において，当時主流だった SVM を用いた手法に大差をつけて優勝したことで有名となり，ディープラーニング研究発展の火付け役となった．現在でも CNN ベースのモデルは多く研究されており，他の手法と組み合わせるといった応用も行われている．

3.3.2　ResNet

ResNet（Residual Network）とは，2015 年 Microsoft の Kaiming He が考案した，152 層という深い層を持つ CNN である．[4]

当時 CNN のモデルは，多層にすることで性能が向上すると考えられていたが，深くしすぎても精度が向上せず，逆に様々な問題が起こることが発見された．主要な問題としては，勾配消失問題があり，勾配がほぼ 0 となってしまい，収束しなくなる問題である．誤差逆伝播法では誤差を最小化するように，勾配を用いて学習を行っていく．その際，出力層から入力層に向かって勾配を乗算していくことになる．シグモイド関数のような活性化関数を使う場合は，勾配の最大値が低く，乗算を重ねるほど勾配が極めて 0 に近くなってしまうのである．この問題の解決策として，バッチ正規化が考案されたが，誤差が大きくなることで精度が落ちてしまう劣化問題が別で発生してしまった．

これらの問題を解決するように開発されたのが ResNet である．ResNet では残差ブロックを用いることで，劣化問題を解決し，モデルのさらなる多層化に成功した．残差ブロックでは，2 から 3 層の畳み込み層に対して，スキップ接続と呼ばれるものを接続する．スキップ接続では，残差ブロックに対する入力の値を出力の前にそのまま加算する．残差ブロックの最終的な出力を $H(x)$，残差ブロック内の出力を $F(x)$，入力を x と置く

と，$H(x) = F(x) + x$ という形になる．

そうすることで，勾配を計算する際に誤差を減衰させることなくそのまま伝えることができるのである．

ResNet は上記の構造を持って CNN のさらなる多層化を実現したが，その構造はシンプルであり，現在を含めて様々な研究のベーシックなモデルとして活用さてれいる．

3.3.3　Simple Baseline

ディープラーニングを用いたキーポイント推定では，一般的に CNN を用いた畳み込みをすることで特徴抽出を行う．しかし，この手法では畳み込みを行うにつれて特徴マップが小さくなることから，位置情報が失われてしまい，元の画像サイズで座標指定を正確にすることが難しいという欠点が存在する．この欠点は姿勢推定分野に往年の課題であり，その欠点を克服するために，数多くの研究が行われ，モデルが開発されてきた．その中のモデルの一つが，2018 年に Microsoft 社が提案した Simple Baseline である．[5]

当時，多様なモデルが研究されていく中で，モデル構造は次第に複雑になっていったという背景が存在する．モデルの複雑化は，比較や有効性検証などの分析をするにあたってそれが困難となるという問題を引き起こす．それを解決するために，Simple Baseline は名前の通りシンプルな構造というコンセプトで開発された．

図 3.2 のように，モデル構造は非常にシンプルである．前半では ResNet を用いて特徴マップを生成し，後半ではそれに対して転地畳み込み（逆畳み込み）を行うことで小さくなった特徴マップを拡大している．

転地畳み込みとは，元となる特徴マップに対し操作を行うことでそれを拡大してから，通常の畳込みを行う手法である．転地畳み込みは既存の手法と違い重みを学習するため，より高い精度が出やすいというメリットが存在する．図 3.3 は転地畳み込みの具体例であり，手順としてはまずストライドという値で指定された数に応じて，元データを拡張する．次にカーネルのサイズから 1 を引いた分だけ周囲を拡張する．最後に通常の畳み込みと同じ計算を行うことで特徴マップを拡張するのである．

Simple Baseline ではストライドを 2，カーネルのサイズを 4 で転地畳み込みを行っている．最後にサイズ 1 のカーネルを用いた畳み込みを行い，ヒートマップを作成することでキーポイントの推定を行うのである．

Simple Baseline は構造の簡略化をしながらも，既存手法を上回る精度を出したことか

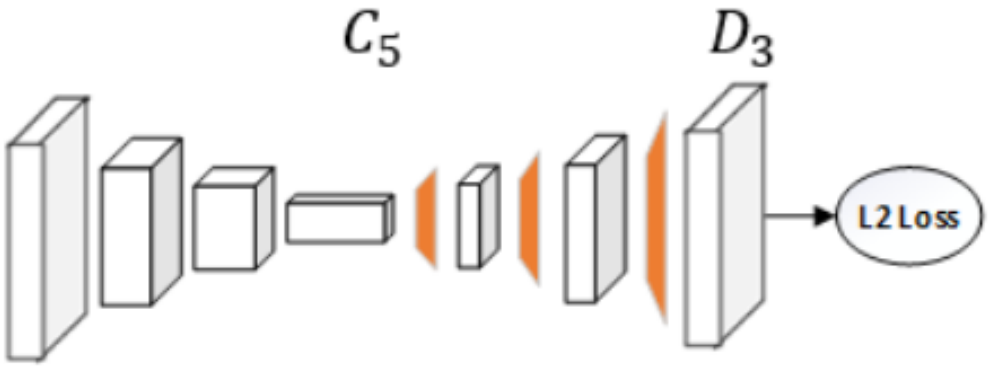

図 3.2:　Simple Baseline のモデル概要．引用元 https://openaccess.thecvf.com/content_ECCV_2018/papers/Bin_Xiao_Simple_Baselines_for_ECCV_2018_paper.pdf

ら，論文では既存手法のように複雑化せずとも同等以上の性能を出すことができると結論付けられている．しかし，新規性を重視しているわけではなく，あくまでベースラインのモデルの提案であり，論文では適切なベースラインや比較の重要性が述べられている．

3.3.4　HRNet

HRNet は，High-Resolution Net の略であり，CVPR19 で発表された姿勢推定モデルのアーキテクチャである．[6] COCO データセットおよび MPII データセットを用いたキーポイント検出タスクで高い精度を記録し，姿勢推定タスクのブレイクスルーとなった．

従来のアーキテクチャでは，Simple Baseline のように，特徴量を一度低解像度に落としてから高解像度化をするという流れが一般的であった．しかし，HRNet は図 3.4 の上段をメインとして，処理全体で高解像度の特徴量を維持した状態で処理を行う．具体的には，入力画像に対して最初に 2 回の畳み込みを行い，それをメインルートとする．また，それに並行して，低解像度で高度な特徴量化を行う．処理途中で異なる解像度の特徴

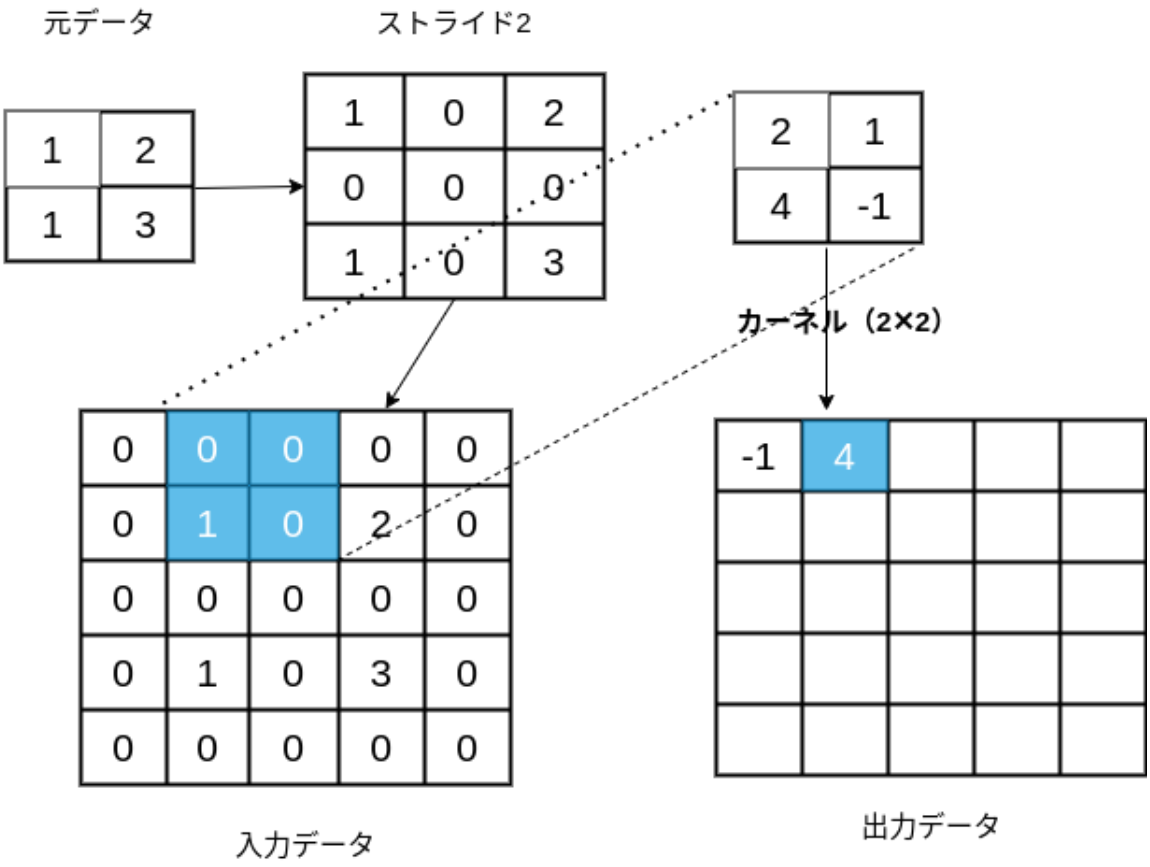

図 3.3: 転地畳み込みの具体例

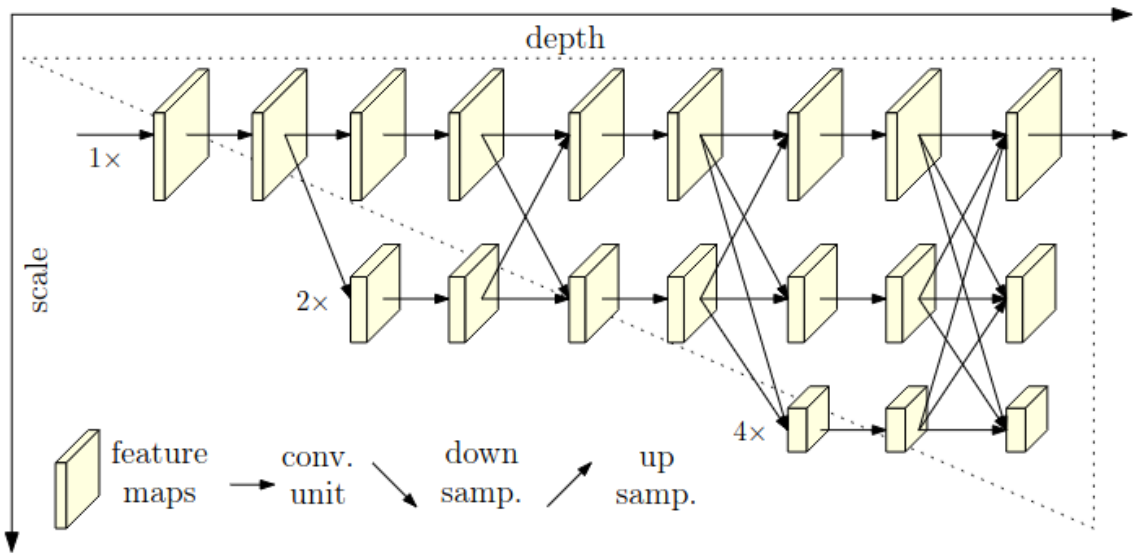

図 3.4: HRNet のモデル概要．引用元 https://openaccess.thecvf.com/content_CVPR_2019/papers/Sun_Deep_High-Resolution_Representation_Learning_for_Human_Pose_Estimation_CVPR_2019_paper.pdf

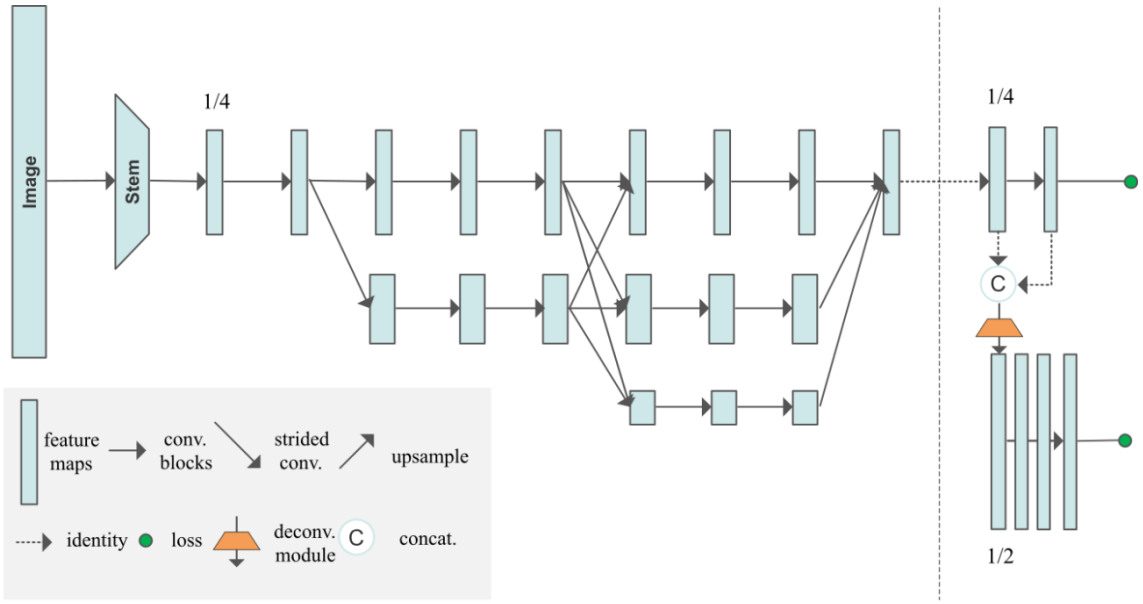

図 3.5:　HigherHRNet のモデル概要．引用元 https://openaccess.thecvf.com/content_CVPR_2020/papers/Cheng_HigherHRNet_Scale-Aware_Representation_Learning_for_Bottom-Up_Human_Pose_Estimation_CVPR_2020_paper.pdf

量何度も融合することで，マルチスケールな特徴量を作成していくことができる．既存研究における課題であった高解像度化を，低解像度の高度な特徴量を高解像度を保った状態で得るのである．

3.3.5　HigherHRNet

HigherHRNet とは，HRNet をベースとした Bottom up 型の姿勢推定モデルである．[7] Bottom up 型アプローチの問題点として，画像内の人間の大小に対応しづらい，つまり対象のサイズの違い弱いにというものが存在する．これは，Top down 型アプローチと違いサイズの変更ができないためである．従来の Bottom up 型アプローチでは，画像サイズそのものを変えることでこの問題点を解決しようとしていたが，ヒートマップの質は依然として十分ではなかった．

HigherHRNet はこの問題点を解決するために，転地畳み込みをする機構を追加した．図 3.5 は HigherHRNet のモデル概要であり，点線から左側が HRNet，右側が論文内で提案された部分である．従来のヒートマップによる姿勢推定に加えて，転地畳み込みを行い解像度を上げたヒートマップでも推定を行うのである．重要なのが，元の解像度と

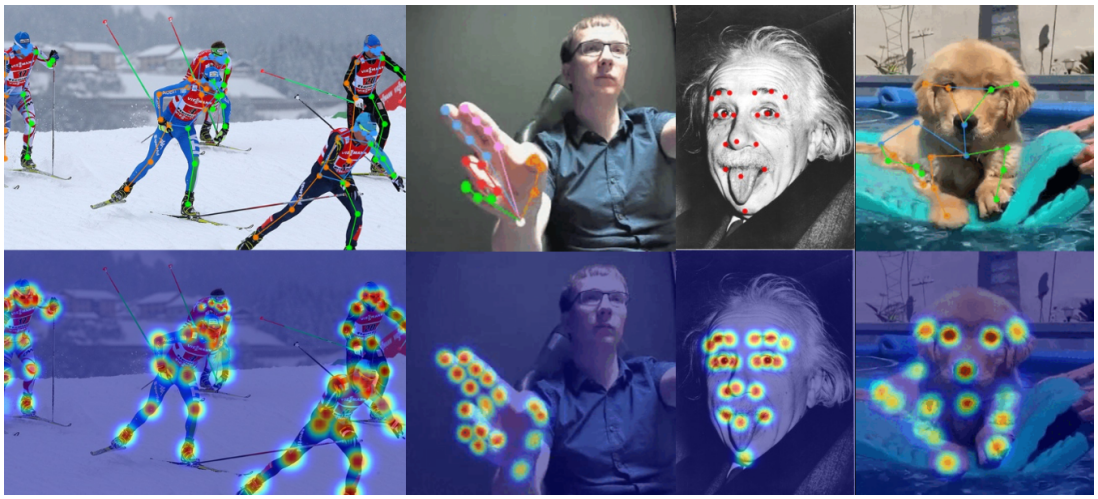

図 3.6:　mmpose の活用例 引用元 https://mmpose.readthedocs.io/en/latest/overview.html

上げた解像度の両方で推定を行うという点である．どちらか一方では，大きいサイズもしくは小さいサイズに特化してしまい，特化していないサイズの推定精度が下がってしまう．そこで，両方からヒートマップを作成し，それを合わせることでサイズの大小に対応したのである．

HigherHRNet は上記の機構が有効に働き，Bottom up 型モデル内で，COCO データセットを用いたテストで最高精度を達成した．また，人が多く写っているようなデータセットでは Top down 型をも上回る精度を記録した．

3.4　MMPose

MMPose とは，OpenMMLab が開発している PyTorch に基づく骨格推定用のオープンソースのツールキットである．商標利用が可能な点や，人間，手，顔，動物の骨格推定といった幅広いタスク用のモデルを使用することが出来る点が特徴である．また，学習済みモデルも多く存在するため，単純に推論を行いたいだけなら画像とモデルをロードするだけでいいというのも利点である．本研究では，MMPose を用いて姿勢推定を行っている．

第 4 章

CLIP:Contrastive Language-Image Pre-training

4.1 概要

CLIP(Contrastive Language-Image Pre-training) とは，OpenAI が開発し，2021 年 2 月に公開された画像と自然言語を結びつけたコンピュータービジョンモデルである．[8] 画像とその画像に関連する文章のペアを大量に用意し，テキストと画像の関連性を学習することで，画像とテキストの両方を埋め込むことが出来る．

画像処理モデルにおける往年の課題として，学習用データ作成のコストが非常に高い点が挙げられる．なぜなら，画像とラベルのペアを用意しなければならないからである．また，従来のモデルでは，予め定められたカテゴリを学習するため，そのカテゴリの分類に最適化され，別のタスクに応用することが難しいという欠点があった．

この問題の解決策として，CLIP ではインターネット上から集めた画像とテキストのペアをデータセットとすることで，ラベル付というコストを削減している．また，ネット上から得られるテキストは，従来のデータセットのように規定のラベルが存在しない．そのため，自然言語を利用することで画像を分類するようなモデルを構築している．

CLIP の特徴として，ユーザがラベルを自由に設定して画像を分類できるというものがある．これは，自由度の高いテキストをラベルとして学習しているためである．また，CLIP は様々なタスクに対して高い汎用性を持ち，画像生成，画像キャプション生成，画像検索，セマンティックセグメンテーションなどの様々な応用に活用されている点も大きな特徴である．

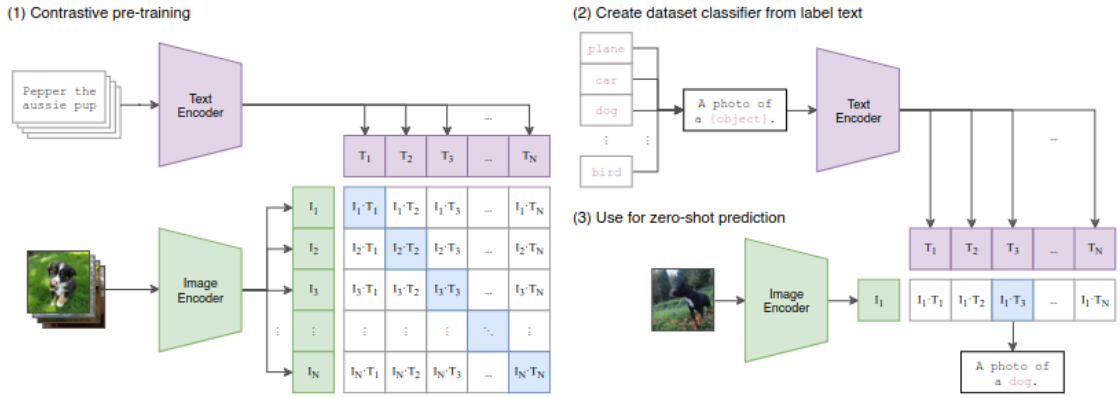

図 4.1: CLIP のモデル構造．引用元 https://openai.com/research/clip

4.2　モデル

CLIP では，Image Encoder として ResNet ベースのモデルを，より効率よく学習しようとしたものとして VisionTransforme を使用している．Text Encoder として Transformer を使用し，Image Encoder と組み合わせて，同時に学習を行う．入力には，複数の画像と複数のテキストをセットにして用いる．出力は，テキストをエンコードして得られるベクトルと，画像をエンコードして得られるベクトル，この二種類のベクトルの内積である．正解となる画像とテキストの組の内積が，1 になるように学習を行う．この時，入力されるテキストから言語間の関係を学習することが出来るのである．CLIP は上記の学習方法で言語間の関係を学習しているため，分類を行う際に自由にラベルを設定することが出来る．また，その際に追加学習を行うことなくゼロショットで，かつ高精度に画像を分類することが出来る．

4.2.1　Transformer

Transformer は，2017 年に Vaswani らによって提案されたモデルである [9]．当時，自然言語モデルで主流であった再帰層や畳み込み層を用いず，Attention 層のみを用いるという点が特徴である．Attention とは，入力の関連度や重要度を計算し，入力のどこに注目するか判別するための仕組みである．Encoder，Decoder 形式のモデルであり，自然言語処理における機械翻訳のためのモデルとして提案されたものである．

Encoder の構造として，特徴的なのは Positional Encoding である．これは，単語が

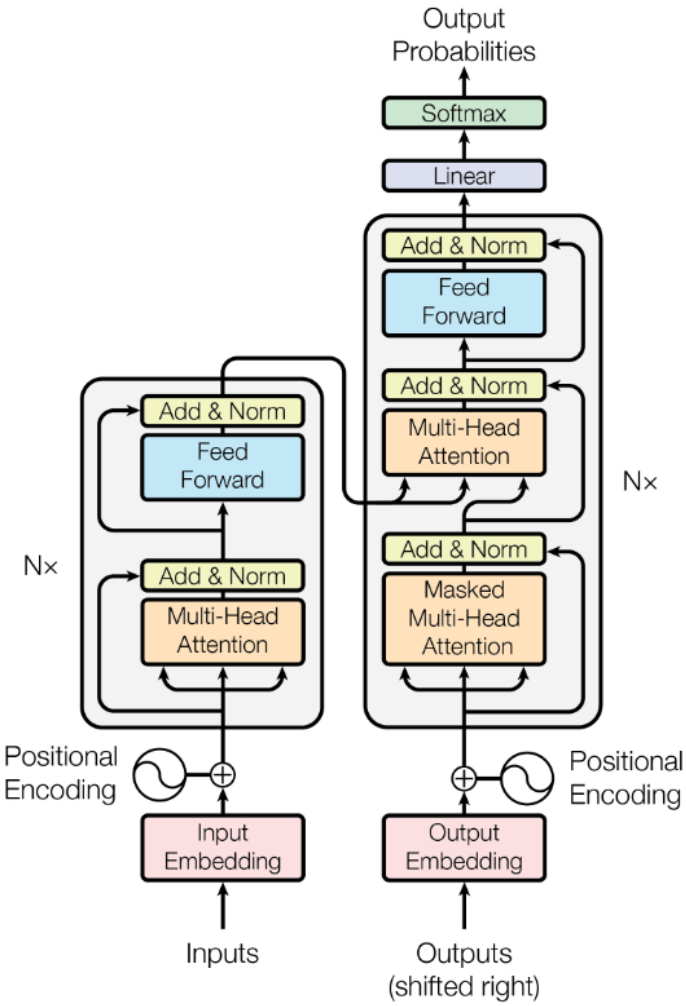

図 4.2: Transformer のモデル概要．引用元 `https://papers.nips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf`

文章内の何番目に当たるかという情報を付与するための機構である．具体的には，以下の計算式を用いてベクトルを算出し，それを足すことで位置情報を付与する．

$$PE(pos, 2i) = sin(\frac{pos}{10000^{\frac{2i}{d_{model}}}}) \tag{4.1}$$

$$PE(pos, 2i+1) = cos(\frac{pos}{10000^{\frac{2i}{d_{model}}}}) \tag{4.2}$$

ここでは，単語の位置を pos，単語埋め込み後のベクトルにおけるインデックスを i，単語埋め込み後のベクトルにおける次元数を d_{model} とする．また，10000 はハイパーパラメータである．これは，付与する位置情報のベクトルが大きくならないかつ，同じ位置で

同じベクトルが付与される必要があるという要件を満たしている．

Encoder 内の Multi-Head Attention では，Self-Attention を計算している．Self-Attention は，入力された文に対する単語間の関連度を計算する機構である．どの単語とどの単語が関連性が高いか，重要度が高いかという情報を持たせた埋め込み表現を出力するのである．Self-Attention は以下のように計算される．

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{4.3}$$

まず，入力単語ごとに，Query，Key，Value という 3 種類のベクトルを算出し，それぞれを Q，K，V とおく．d_k は Query と Key の次元数である．$softmax(\frac{QK^T}{\sqrt{d_k}})$ では，Query と Key の内積を取ることで Attention weight と呼ばれる単語間の類似度を計算している．その Attention weight と Value を用いて単語の特徴量を得るのである．つまり，類似度の高い単語の Value は多めに，類似度の低い単語の Value は少なめに取得するという意味である．

Decoder について，Masked Multi-Head Attention は学習時に，入力となる翻訳後の文章に対して，制限をかけた状態で Self-Attention を行う機構である．予測したい単語の入力に，予測したい単語そのものやその後の情報を与えないようにマスクしているのである．次に Encoder から Key と Value を受け取り，現在の特徴量を Query として Attention を行う．このように翻訳前と後の文章に対して，どこに注目すべきか，どの単語が対応していくかを学習していくのである．

Transformer は，2017 年当時に主流であった CNN や RNN を全く使わない構造でありながら，それらを凌ぐ性能を持っていたことから，現在でも開発されるモデルのベースとなることが多いモデルである．

4.2.2　VisionTransformer

VisionTransformer [10] は Google が 2020 年に発表した Transformer ベースのモデルである．前提として，Transformer は連続するデータの処理を得意としているため，画像処理の分野ではあまり使われていなかった．なぜなら，自然言語処理であれば単語間の関係を学習することが大きな強みと考えられていたが，画像処理においてはピクセル同士の関係を明らかにしてもあまり役立たないと考えられていたからである．画像のピクセル間で Attention の計算を行えば，計算量は膨大になり，しかしその計算量に見合った成果は得られなかったのである．

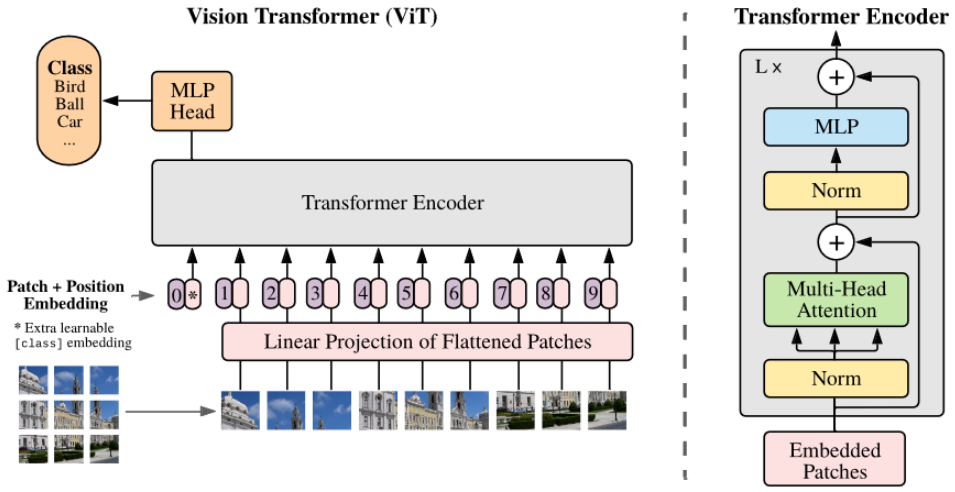

図 4.3: VisionTransformer のモデル概要．引用元 `https://openreview.net/pdf?id=YicbFdNTTy`

VisionTransformer は画像を扱うモデルでありながら，完全に畳み込みを用いずに，また上記の問題点を克服することで，様々なタスクで SoTA 達成した．

VisionTransformer の重要な点として，画像パッチを単語のように扱うというものがある．入力された画像を $x \in R^{H \times W \times C}$ とする．(H, W) は画像の解像度，C はチャネル数 (入力画像の枚数) である．VisionTransformer では，画像を入力する前に画像パッチ $x_p \in R^{N \times (P^2 \cdot C)}$ に変形する．$N = \frac{HW}{P^2}$ であり，これは各パッチの個数である．(P, P) は各パッチの解像度である．

これらのパッチに分けられた画像に対して, Patch Embedding と Position Embedding を行う．Patch Embedding とは，各パッチをベクトル化し，線形射影を行う工程である．Position Embedding とは，各パッチが画像のどこに当たるかを識別するための位置情報を付与する工程である．つまり，これらの処理は，Transformer を画像に用いる際の問題点を解消するように設計されたものである．

上記の工程を画像に施した後に, Transformer Encoder へ入力を行う．Transformer の構造としては, は Multihead Self-Attention ブロックと Multi Layer Perceptron（MLP）が交互に用いられている．また，これらのブロックの前には Layer Normalization が，ブロックの後には残差接続が適用されている．Multihead Self-Attention は，Transformer で記述したものと同様に，パッチごとに Query，Key，Value を用いて関係を学習する．

Multi Layer Perceptron は，画像分類を行う層である．

VisionTransformer は計算コストという面でも優れており，論文中では，巨大なデータセット JFT-300M で事前学習を行ったとき，SoTA を上回る性能を得ながら，計算コストを約 6.6 まで削減することができたと述べられている．また，Transformer を扱う利点として，CNN よりもモデルの拡張がしやすいため，事前学習データセットとモデルをさらに大きくすることでさらに性能が向上すると示されている．

4.3　実験

論文中の実験では，主にゼロショット分類について行っている．ゼロショット分類とは，学習を行っていないラベルについて画像分類を行うタスクであり，未学習のデータセットを用いた画像分類である．実験を行う際の比較対象となるモデルは ImageNet で事前学習を行い，各データセットでファインチューニングを行った教師あり学習済みの ResNet-50 である．

図 4.4 は，27 個のデータセットを用いた比較実験を行った結果である．27 個の内 16 個のデータセットが教師あり学習をしたモデルをゼロショットで上回った．STL10 はラベル付された画像が少なく，またラベルなしデータが多いという特徴を持つ画像認識用のデータセットであり，最も精度が高い結果となった．Country211 は，画像が撮られた国がどこかを当てるデータセットであり，これの精度が高いのはやはり言語間の関係を学習できているためである．また Kinetics700 や UCF101 といった，動画から切り取った画像データセットはラベルが動詞であり，同じ理由で CLIP が精度を大きく上回った．対して，衛生の画像を分類するデータセットである EuroSAT や，リンパ節の画像を分類するデータセットである PatchCamelyon といった，専門的な知識が必要なデータセットでは教師あり学習モデルに大きく劣る結果となった．

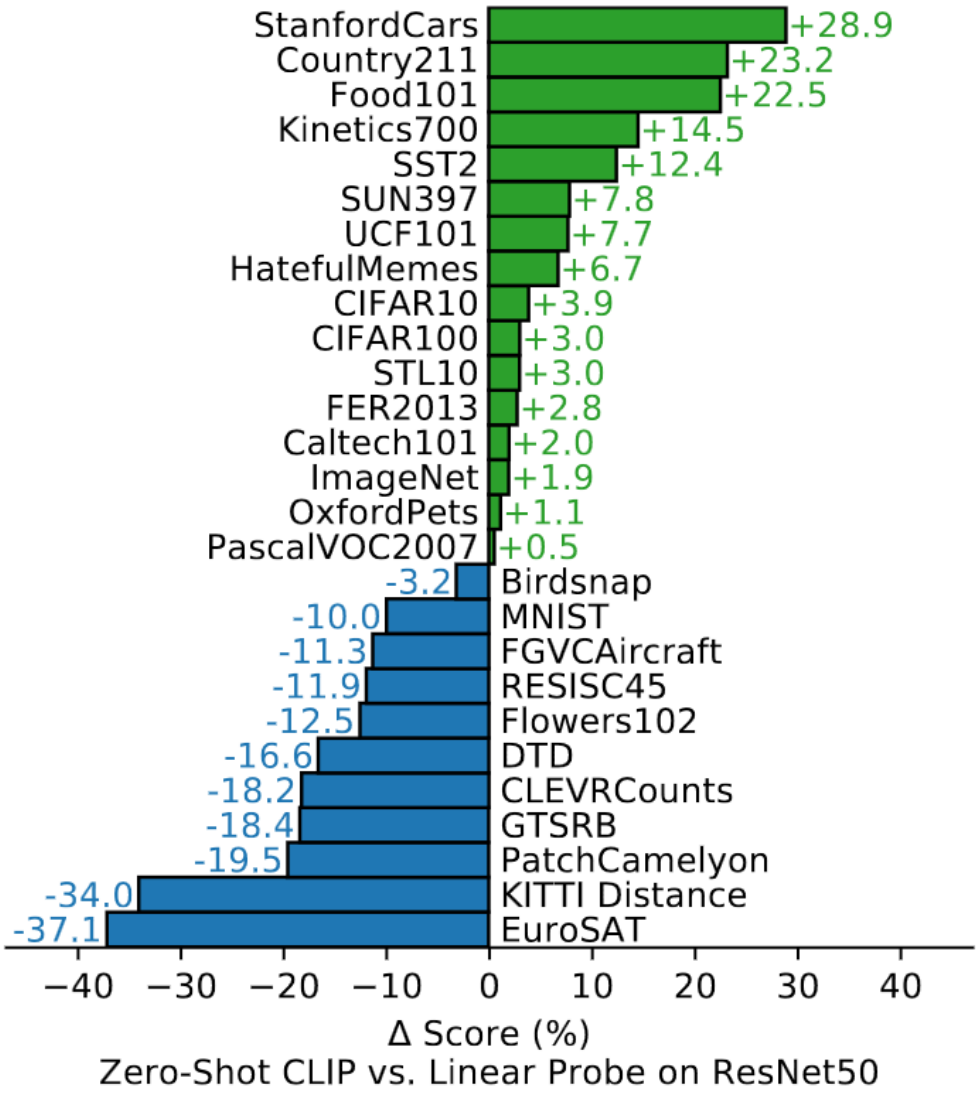

図 4.4: CLIP と ResNet-50 の比較実験結果 引用元 `https://arxiv.org/pdf/2103.00020.pdf`

第 5 章

提案手法

CLIP は画像と自然言語を結びつけたコンピュータービジョンモデルである．人物やその周囲の物体，背景といった様々な情報が CLIP を使うことで得られるため，大まかな動作を推定し，似ている動作の違いを姿勢の違い，つまり姿勢推定で判別することが出来るというアイデアを元にし本研究では，CLIP と姿勢推定を用いた画像内人物の動作推定という手法を提案する．

単一画像に対して CLIP を用いて画像処理を行い，周囲の状況や人と物体に関する特徴量を取得する．更にそこに MMPose による姿勢推定を組み合わせることで，画像内人物の骨格情報を取得する．これら二つのベクトルを組み合わせることで，人体の姿勢情報と画像全体の情報を融合させ，動作推定を行う．

第 6 章

実験

6.1 実験設定

本実験では，CLIP と MMPose を用いたモデルを作成し，画像内人物に対して動作名を予測することで精度を計測した．

モデルは以下の三通りを作成した．

- CLIP のみを用いたモデル
- MMPose のみを用いたモデル
- CLIP と MMPose を用いたモデル

提案手法は CLIP と MMPose を用いたモデルであり，ベースラインは CLIP のみを用いたモデルである．比較用として MMPose のみを用いたモデルも作成した．これら 3 通りのモデルを比較することで提案手法の有効性及び各パーツがどれほど精度に寄与しているかを確認した．

評価手法には，単純な正解率を用い，それを精度とした．正解率が高ければ精度が高いモデルであり，正解率が低ければ精度が低いモデルである．尚，本研究で用いるデータセットには複数のラベルがアノテーションされているため，回答が完全一致した場合のみを正解という扱いにし，部分点は用いていない．

複数の実験を複数のモデルを作成して行ったが，設定は全てにおいて，

- エポック数 50
- バッチサイズ 15

- 学習率 0.0001
- 最適化手法 Adam
- 損失関数 BCEWithLogitsLoss

である．

6.2　実験データ

実験データには V-COCO [11] を改変して使用した．V-COCO(Verbs in COCO) は本来, Human-Object Interaction Detection のために MS-COCO [12] から人物が写っている画像を抜き出し，アノテーションされて構築されたデータセットである．MS-COCO は物体検出やキャプション生成に用いることができる大規模なデータセットである．V-COCO はアノテーションされた動作名ごとに分けられた dict 型を，list 化したデータセットである．以下に，dict のキーとその説明を示す．

■image-id　画像ごとに割り振られた id の list である．MS-COCO で用いられる id と同様のものであり，MS-COCO の API を用いれば画像名や，画像に対してアノテーションされたデータの id を検索することができる．

■ann-id　人ごとに割り振られたアノテーション id のリストである．image-id と同様に MS-COCO で用いられているものと同一である．MS-COCO の API を用いれば，アノテーションの内容やどの画像にアノテーションされているかを検索することができる．

■action-name　hold, stand, sit, ride, walk, look, hit, eat, jump, lay, talk-on-phone, carry, throw, catch, cut, run, work-on-computer, ski, surf, skateboard, smile, drink, kick，point，read，snowboard の計 26 種類の動作ラベルの内一つ割り振られている．

■role-name　その動作にものが必要かどうかが入力されている．action-name が stand である dict 内であれば，立つのにものは必要ではないため agent とだけ入力されているが，action-name が kick である dict 内であれば，蹴る対象が写っているため agent と obj と入力されている．

■role-object-id　role-name で指定された agent や obj のアノテーション id が割り振られている．

■include　写っている，動作に関係する物の物体名のリストである．例として，action-name が drink であれば，wine glass や bottle などがリスト内に存在する．

この中で，本研究に必要な情報である image-id，ann-id，action-name のみを抜き出し，アノテーション id をキーとした dict 型のデータセットを作成した．また，人物ごとに動作の学習をするには，その人物を抜き出した画像を入力する必要がある．そこで，MS-COCO にアノテーションされていたバウンディングボックスを用いて切り取った画像を入力とするようにした．そのため，V-COCO から作成したデータセットにバウンディングボックスを付与したデータセットが，本研究で使用したデータセットである．

データの総数は訓練データ 3896，テストデータ 7674 である．尚，アノテーションされた動作名は複数個であり，本研究では他ラベル分類を行う．訓練データにアノテーションされた動作名の総数は 11325 個，テストデータにアノテーションされた動作名の総数は 22070 個であり，1 人の人物に対して，平均して約 3 個の動作名が付与されている．

表 6.1 はアノテーションされた各種動作の総数である．動作には含有の関係があり，stand や hold といった抽象度の高い動作は多くアノテーションされているが，point や read といった，抽象度の低い動作は少ないという特徴がある．

6.3　モデル

本研究で使用した事前学習済みモデルは以下の通りである．

- CLIP

  ViT-B/32 を使用した．これは Image Encoder に VisionTransformer を使用し，画像パッチを 32*32 で学習を行ったモデルである．
- MMPose

  MS-COCO を用いて事前学習された HigherHRNet を使用した．HigherHRNet は Bottom Up 型であり，人物の検出を行わないため他の手法との差別化ができると考えたためである．

図 6.1 は，提案手法モデルの概略である．

モデルの学習の流れは以下の通りである．

1. CLIP を用いて画像の特徴量を取得する．

動作名	訓練データ	テストデータ
hold	1838	3608
stand	2150	4118
sit	935	1916
ride	252	556
walk	300	597
look	1770	3347
hit	164	349
eat	301	521
jump	319	635
lay	243	387
talk-on-phone	167	285
carry	236	472
throw	159	244
catch	152	246
cut	127	269
run	305	687
work-on-computer	210	410
ski	190	424
surf	239	486
skateboard	220	417
smile	678	1415
drink	63	82
kick	65	180
point	15	31
read	51	111
snowboard	176	277

表 6.1: データセットの内容

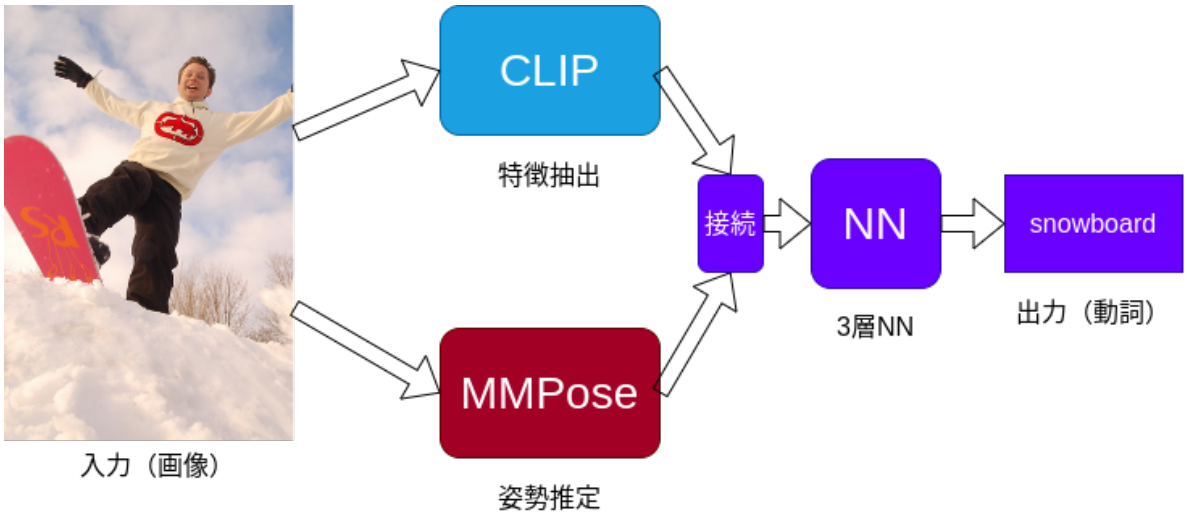

図 6.1: 動作推定モデルの概略図

出力は 512 次元のベクトルとなる．

2. バウンディングボックスを用いて人を切り出し，MMPose から画像内人物のキーポイントを取得する．

   出力は 17*3 次元のベクトルとなる．

3. 2 つのベクトルを接続する．

   まず，MMPose 側の出力を平坦化し，51 次元のベクトルに変形する．その後，CLIP 側の出力の最後尾にそのまま結合し．563 次元のベクトルに合成する．

4. 3 層 NN に入力する．

   3 層目で 26 次元にすることで，26 種類の動詞に割り振る．

5. 三層 NN の出力から，値を高いものを出力とする．

   このとき，正解となるラベルの個数が何個かを取得し，その個数分上位の値を抜き出すことで，動詞の出力を行う．

6. 最適化を行う．

   モデルで学習を行うのは NN 部分だけであり，CLIP と MMPose は行わない．

2 番目でバウンディングボックスを用いてキーポイントを取得する理由は，学習する人物のキーポイントと学習ラベルの対応をするためである．MMPose の出力は，画像内に複数人の人物が写っていた場合，その人数分キーポイントの出力を行う．そのため，元画像そのまま入力を行う場合，どのキーポイントが学習を行いたい人物のものか判別することが難しいのである．そこで，バウンディングボックスを用いて画像を切り取ることで，

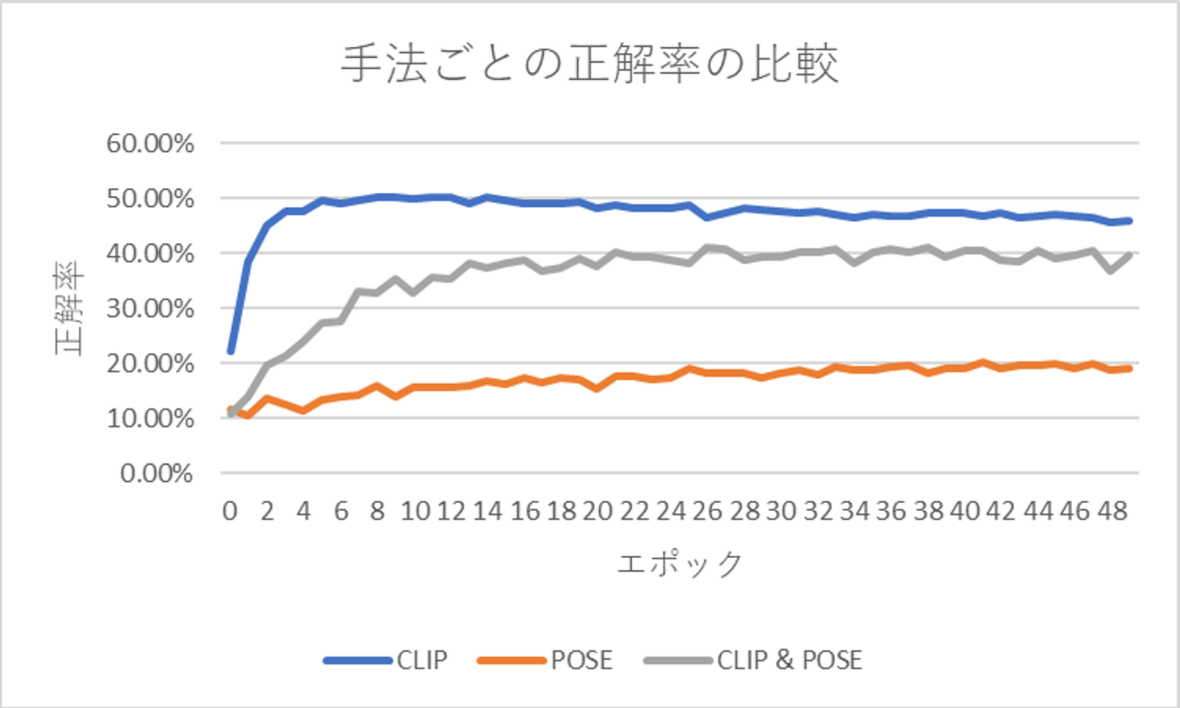

図 6.2: 動作推定結果の正解率の比較

出力されるキーポイントを限定することができる．尚，MMPose の出力にはキーポイントに加えて score という値が存在する．これはキーポイントがどれだけ正確であるかを表す値である．バウンディングボックスで切り取った場合でも複数人分のキーポイントが出力される場合は，この score の値が一番大きいキーポイントを扱うこととしている．

6.4　実験結果

6.4.1　実験 1

作成した 3 通りのモデルにより実験結果は，図 6.2 のようになった．予想とは異なり，CLIP のみを使用したモデルが，最高精度が 50.40% と最も精度が高く，ついで提案手法が 41.18%，最も精度が低いモデルが MMPose のみを用いたモデルであり，最高精度が 20.19% だった．

6.4.2　実験 2

実験 1 の結果を踏まえて，提案手法は精度を下げてしまうことがわかった．しかし，CLIP のみを使ったモデルと提案手法のモデルとでは，CLIP に入力する画像がバウン

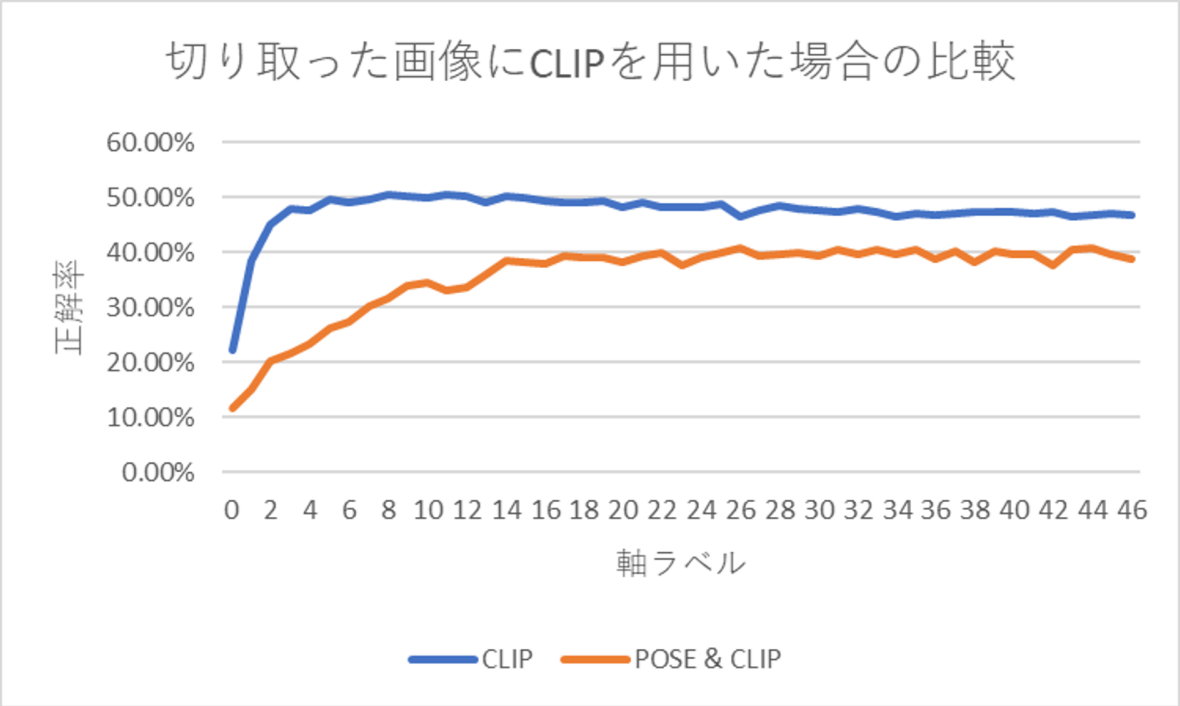

図 6.3: 切り取った画像に CLIP を用いた場合の正解率の比較

ディングボックスで切り取られているか否かでことなっている．そこで，提案手法のモデル内の CLIP に対してもバウンディングボックスで切り取られた画像を入力するように改良し，再度比較実験を行った．実験結果は図 6.3 のようになった．

結果としては，提案手法モデルの精度は実験 1 のものとほとんど変化がなく，最高精度は 40.83% であった．

6.4.3　実験 3

上記の結果を踏まえて，提案手法のままでは MMPose 側のベクトルがノイズとなってしまい精度が下がっているという予想を立てた．そこで，MMPose の出力に含まれる score という値に着目した．

テストデータには，上半身のみが写っていたり，持っているもので体の大部分が隠れてしまっている画像が存在する．それらの画像は score の値が低く，動作推定をする上で，MMPose 側のベクトルがノイズとなっている原因ではないかと考えた．

そこで，テストデータに score の制限をかけてテストを行った．score の値は 0.5 以上とし，テストデータ数は 1557 まで減少した．

実験結果は，図 6.4 のようになった．CLIP のみを使ったモデルの最高精度が 52.22%

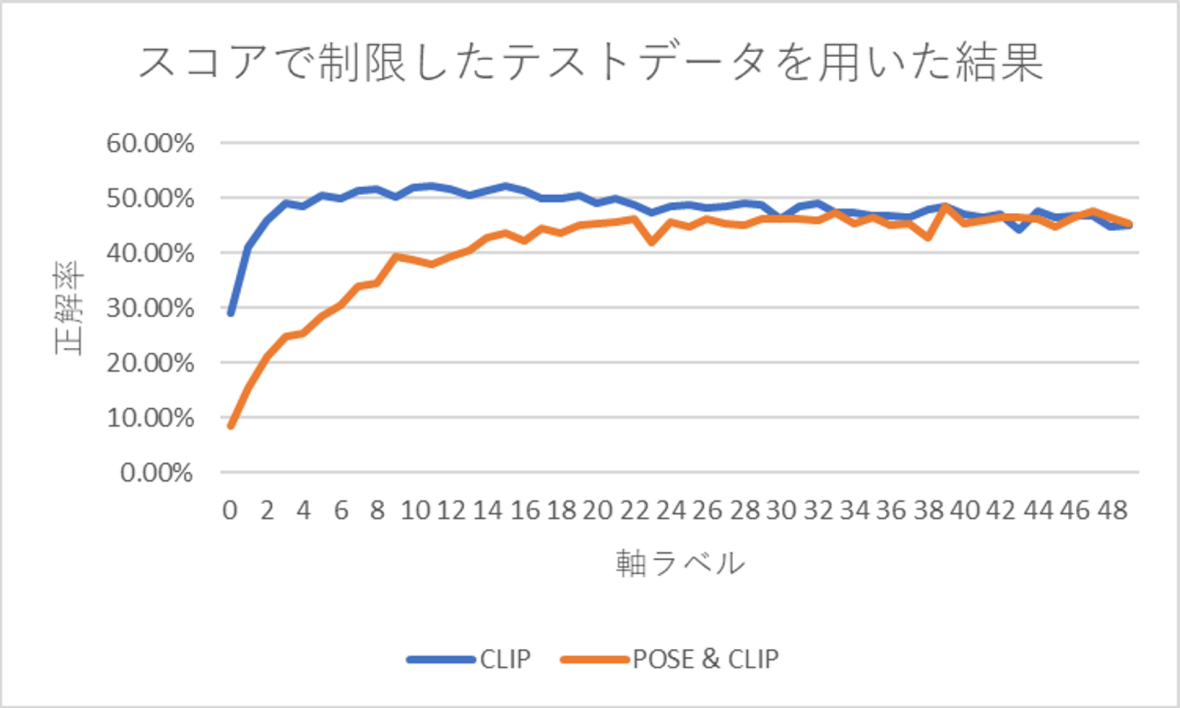

図 6.4: score で制限をしたテストデータを用いた正解率の比較

であるのに対し，提案手法のモデルの最高精度は 48.36% であった．

依然として，CLIP のみを使ったモデルのほうが高い精度を出しているが，実験 1 や実験 2 の結果よりも提案手法のモデルの精度が上昇していることが確認できた．また，精度の上昇率に関して言えば，CLIP のみを使ったモデルが 1.82% の上昇だったのに対し，提案手法のモデルは 7.18% の上昇だった．

6.4.4　実験 4

実験 3 から，score の値が高い画像については MMPose の特徴量がノイズになりにくいという予想を立てた．そこで，訓練データにも score の値で制限をかけ，精度の変化を調査した．

訓練データも，テストデータと同様に score を 0.5 以上で制限をかけたところ，総数は 777 個となった．その中でも，データの個数が一桁である，cut，drink，point，read の動作が含まれる人物を訓練データ及びテストデータから排除し，765 個のデータを訓練データとして学習を行い，1541 個のテストデータで精度の比較を行った．実験結果は，図 6.5 のようになった．CLIP のみを使ったモデルの最高精度が 47.11% であるのに対し，提案手法のモデルの最高精度は 40.04% となった．これまでの実験と比べ，訓練データの

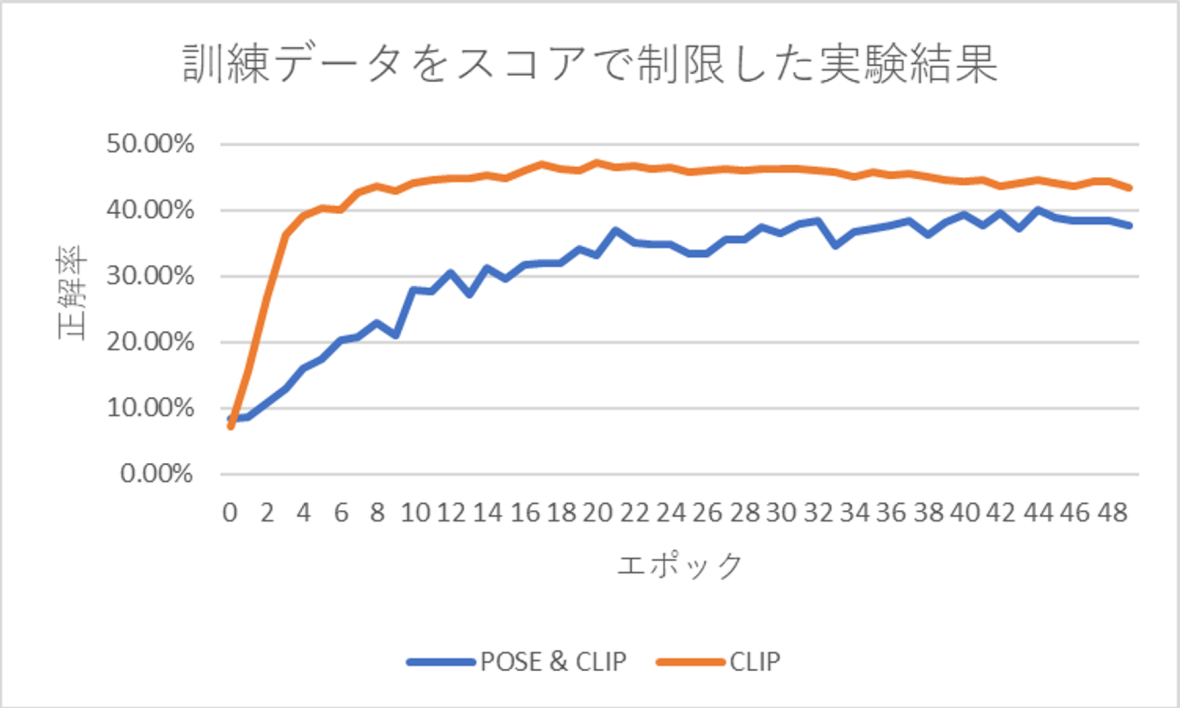

図 6.5: score で制限した訓練データで学習したモデルの正解率の比較

総数が減ったことによる全体的な精度の低下が確認できた．また，実験 3 に比べて提案手法のモデルが大きく精度を下落させる結果となった．

第 7 章

考察

実験 1 から実験 4，すべての実験で提案手法モデルは CLIP のみを使ったモデルより精度が低いという結果が得られた．これには様々な理由が考えられる．

■特徴量の有効性

実験 1 から実験 4，すべての実験の結果に共通することとして，提案手法のほうが精度の収束が遅いという事実が挙げられる．

CLIP のみを使ったモデルの場合は，およそ 3 エポックである程度精度が上昇し，15 エポック頃に最高精度を迎え，その後は過学習の影響で低下していく．それに対して，提案手法のモデルは，精度の上昇が緩やかであり，20 エポックほどで上昇が停止しても過学習の影響で下がることがなく，多少の上下をしながら横ばいに続く．CLIP 側のグラフの形状は，機械学習モデルにおける正常な学習をした場合に近い形をしている．つまり，適切に学習ができているのである．対して，提案手法のモデルは上昇と下落を繰り返し，学習しきれていないことが見て取れる．つまり，提案手法で追加される特徴量がノイズになり学習を妨げていることは確実である．

ただ，姿勢推定で得られる特徴量全てがが完全にノイズであり，学習の妨げになっているというわけではないことは確かである．本研究の実験は 26 種類のラベルの，平均 3 個の他ラベル分類であり，完全一致のみを正解としている．実験 1 の結果を見ればわかるとおり，MMPose のみを用いた手法は最も精度が低くはあれど，ランダムと同程度の精度ということはないのである．つまり，多少の学習はできているのである．

これらのことから，提案手法のモデルが精度を下げた理由は，CLIP から得られる特徴量の有効性が想像以上に高く，姿勢推定と差が付きすぎてしまったため，結果としてノイ

動作名	テストデータ	CLIP	提案手法
hold	626	264	206
stand	1078	551	475
sit	136	28	21
ride	50	7	11
walk	66	11	13
look	889	475	387
hit	192	95	78
eat	15	3	1
jump	256	124	119
lay	17	2	0
talk-on-phone	23	2	1
carry	49	9	6
throw	139	50	25
catch	99	15	6
run	408	183	164
work-on-computer	17	1	2
ski	115	78	60
surf	160	112	97
skateboard	160	112	92
smile	175	42	40
kick	127	78	73
snowboard	63	26	19

表 7.1: 実験 3 における正解ラベル数

ズのような働きをしてしまったのではないかと考察している．

■姿勢推定という手法の有効性

姿勢推定を取り入れた理由は，細かな動作を分類出来るようにするためである．しかし，本研究で用いた V-COCO データセットでは，26 種類の動作で割り振られており，

その動作はすべてが似ているというわけではない．もちろん sit と ride といったように似ている動作も存在するが，明確に異なる動作も少なくない．つまり姿勢推定が必要になるような動作の分類は少なく，CLIP による画像処理で事足りる場合が多かったのである．そういった場合は，姿勢推定という情報がノイズになってしまうと考えられる．

表 7.1 は実験 3 で作成したモデルを用いた，正解ラベル数の比較である．ほとんどのラベルで，CLIP のみを使ったモデルが提案手法のモデルを上回っていることが確認できた．また，eat や talk-on-phone など，何を扱っているか，何が写っているかで明確に推測できる画像は，両モデルともに周辺の情報が得られないため，ほとんど正解できていない．

提案手法のモデルが，CLIP のみを使ったモデルよりも正解数が多い動作は，ride，walk，work-on-computer の 3 つのみである．これらの正解数の差は極めて少ないが，姿勢推定の情報を使うことで正解できた可能性が高い．特に，ride と walk に関しては，目論見通りに提案手法がうまく働いた具体例であると言える可能性がある．

■データセット

V-COCO データセットの多くが姿勢推定に適していない画像であったことは，精度が下がってしまった要因の一つであると考えられる．訓練データ全体の score は，平均して 0.35 であり，低い数値である．つまり，何かに隠れてしまっていたり，上半身しか写っていないような画像がデータセット内に多く存在したのである．実際，実験 2 と実験 3 の結果を比べると，明らかに精度の差が縮まっているため，score の値が精度に直結することは明白である．また，smile や look といった動作は，姿勢とはほとんど関わりのない動作である．そういった点でも，V-COCO データセットは適していなかったと考えられる．

総じて，提案手法はほとんどの場合において，動作推定に対して有効的ではないということが確認された．とりわけ，score の低い画像はノイズとなってしまい，精度を下げる要因になってしまうことがわかった．

しかし，提案手法が僅かだが正解数を上回った動作も確認することができた．よって，姿勢推定に適した画像をつかい，またそれに適したラベルが付与された，十分な量のデータセットがあれば，提案手法が CLIP のみを用いたモデルを超える可能性がある．

また，キーポイントの扱いについても改良の余地があると考えられる．本研究では，得

られた特徴量を一次元化し，そのまま CLIP の特徴量に接続していた．キーポイントとは即ち関節点座標そのものであるため，学習に適さない形であった可能性がある．キーポイントを適した形に変換する NN を追加することで，提案手法の精度は上昇する可能性がある．

第 8 章

結論

本研究では，CLIP と姿勢推定を組み合わせた動作推定の手法を提案し，提案手法を用いることで詳細に単一画像を認識できているかを調査した．

実験では，提案手法と，画像処理または姿勢推定のみを使用したモデルを作成し各モデルの精度を比較した．また，モデルの仕様を変更したり，score の数値で使用するデータセットを制限したりすることで，様々な状況下の提案手法の効果を調査した．しかし結果は理想に反し，全ての実験結果で，提案手法では寧ろ動作推定の精度を下げてしまうことが示された．

実験の考察では，精度が下がった要因として，score の低い姿勢推定結果がノイズになってしまう点と，CLIP から得られる特徴量の有効性が想像以上に高かった点、提案手法と動作のラベルのミスマッチ，つまりデータセットが適していなかった点が挙げられた．また，姿勢推定で得られたキーポイントの扱いが適していなかったことに関しても，精度の下がった要因である可能性があると結論付けられた．

しかし，一部動作ではあるが，提案手法が精度を上回った例も確認された．そのため，局所的ではあるが，提案手法が完全にノイズになっているというわけではなく，目論見通りに働く場合もあることがわかった．そのため，動作ラベルやモデル構造を改良して実験を行えば精度の向上が期待できる．

謝辞

謝辞は提出する直前に書くのがマナーらしい

参考文献

[1] 智香子高崎, あつ子竹房, 秀基中田, 正人小口. 姿勢推定ライブラリ openpose を用いた機械学習による動作識別手法の比較. 第 81 回全国大会講演論文集, Vol. 2019, No. 1, pp. 275–276, 02 2019.

[2] Yuya Yoshikawa, Jiaqing Lin, and Akikazu Takeuchi. Stair actions: A video dataset of everyday home actions. *arXiv preprint arXiv:1804.04326*, 2018.

[3] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, Vol. 25. Curran Associates, Inc., 2012.

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

[5] Bin Xiao, Haiping Wu, and Yichen Wei. Simple baselines for human pose estimation and tracking. In *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018.

[6] Ke Sun, Bin Xiao, Dong Liu, and Jingdong Wang. Deep high-resolution representation learning for human pose estimation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 5693–5703, 2019.

[7] Bowen Cheng, Bin Xiao, Jingdong Wang, Honghui Shi, Thomas S. Huang, and Lei Zhang. Higherhrnet: Scale-aware representation learning for bottom-up human pose estimation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[8] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh,

Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. *CoRR*, Vol. abs/2103.00020, , 2021.

[9] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, Vol. 30. Curran Associates, Inc., 2017.

[10] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021.

[11] Saurabh Gupta and Jitendra Malik. Visual semantic role labeling. *arXiv preprint arXiv:1505.04474*, 2015.

[12] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *Computer Vision–ECCV 2014*, pp. 740–755. Springer, 2014.

付録

A　提案手法で使用したプログラムリスト

データセットを学習用に整形するプログラムを A.1 に示す．尚，実験によって様々なデータセットを作成したが，代表して，実験 1 で用いた提案手法用の訓練データを作成するプログラムを示す．

ソースコード A.1: withbboxdatareform.py

```
1
2 import matplotlib
3 import matplotlib.pyplot as plt
4 import __init__
5 import vsrl_utils as vu
6 import numpy as np
7 import torch
8 import clip
9 import matplotlib.pyplot as plt
10 from PIL import Image
11 import glob
12 import os
13 import numpy as np
14 from torch.utils.data import DataLoader
15 from torch import nn
16 from torch.nn import functional as F
17 from torch import optim
18
19
20 coco = vu.load_coco()
21
22 #Load the VCOCO annotations for vcoco_train image set
23 vcoco_all = vu.load_vcoco('vcoco_train')
```

```
#画像ロードようのディレクトリ
file_base_dir = os.path.expanduser("~/clip_prefix_caption/
    CLIP_prefix_caption/ClipCap-for-Japanese/train2014/")


device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

#ごとに骨格推定したやつをロード bbox
import pickle
f = open("./trainimage_to_mmpose_cutbybbox_plus20.pkl","rb")
image_to_pose = pickle.load(f)
f.close()


#画像と動作の辞書をロード id
import pickle
f = open("./simple_trainimage_to_clip.pkl","rb")
train_clip = pickle.load(f)
f.close()

#アノテーションと動詞の組み合わせをロード id
import pickle
f = open("./ann_dic_withbbox.pkl","rb")
train_verb = pickle.load(f)
f.close()

#骨格推定データに動作データをくっつけるよ
for d in image_to_pose:
    image_to_pose[d]['image_name'] = train_verb[d]['image_name']
    image_to_pose[d]['verb'] = train_verb[d]['verb']


#データを成型するよ　返り値はのリスト tensor とラベルのリスト()
    tensoronehot
#は tensor512(clip)+51(mmpose)で次元 563

def data_change(d_clip, d_mmpose):
    verb_index = ['hold', 'stand', 'sit', 'ride', 'walk', 'look', 'hit
        ', 'eat', 'jump', 'lay', 'talk_on_phone', 'carry', 'throw', '
        catch', 'cut', 'run', 'work_on_computer', 'ski', 'surf', '
        skateboard', 'smile', 'drink', 'kick', 'point', 'read', '
```

```
        snowboard']
    K = len(d_clip)
    perfect_data = np.empty((0, 563))#とをあわせた型番 clipmmpose
    perfect_label = np.zeros((len(d_mmpose), len(verb_index)))#データ
        数分，のワンホット行列を作る 26
    num = 0
    label_index = []
    for s,v in enumerate(d_mmpose):#で仕切られているを基準にする
        anotationidd_mmpose はアノテーション vid
        if 'score' in d_mmpose[v].keys() and d_mmpose[v]['score'] >
            0:
            perfect_v = []
            for vs in d_mmpose[v]['verb']:#のリストを取り出す verb
                index_num = verb_index.index(vs)#動作のインデックス番号
                    を取得
                perfect_v.append(index_num)
            i = d_mmpose[v]['image_name']#ファイル名を得よう
            m_vec = d_mmpose[v]['keypoints'].flatten()#側のを一次元に
                mmposearray
            c_vec = d_clip[i][0].to('cpu').detach().numpy().copy() #
                側のを抜き出しに
                cliptensornumpy
            perfect_vec = np.append(c_vec, m_vec)#の連結 numpy
            perfect_data = np.append(perfect_data, np.array([
                perfect_vec]), axis=0)#多次元方向に
                追加
            for ind in perfect_v:#
                perfect_label[s][ind] = 1
            #num += 1
        else:
            label_index.append(s)
    print(label_index)
    gati_perfect_data = torch.from_numpy(perfect_data).to(device,
        torch.float)#化
        tensor
    gati_perfect_label = torch.from_numpy(np.delete(perfect_label,
        label_index, 0)).to(device, torch.float)
    return gati_perfect_data, gati_perfect_label


data, label = data_change(train_clip,image_to_pose)
```

学習を行い，エポックごとにモデルを保存する Python のソースコードを A.2 に示す．なお，A.1 と同様に実験 1 で提案手法に用いたプログラムである．

ソースコード A.2: train-cutbybbox.py

```
import matplotlib
import matplotlib.pyplot as plt

import __init__
import vsrl_utils as vu
import numpy as np

import torch
import clip
import matplotlib.pyplot as plt
from PIL import Image
import glob
import os
import numpy as np
from torch.utils.data import DataLoader
from torch import nn
from torch.nn import functional as F
from torch import optim
import withbboxdatareform
import make_train_cutclip_andpose
from torch.autograd import Variable

import pickle

batch_size = 15 #バッチサイズを管理
coco = vu.load_coco()

#Load the VCOCO annotations for vcoco_train image set
vcoco_all = vu.load_vcoco('vcoco_train')
#画像ロードようのディレクトリ
file_base_dir = os.path.expanduser("~/clip_prefix_caption/
    CLIP_prefix_caption/ClipCap-for-Japanese/train2014/")

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')


#これは切り取っていないを使う方 clip
perfect_data = withbboxdatareform.data
perfect_label = withbboxdatareform.label

```

```
40
41
42 #切り取ったほう
43 #perfect_data = make_train_cutclip_andpose.data
44 #perfect_label = make_train_cutclip_andpose.label
45
46
47
48 #を訓練とに分ける datasetval
49 dataset = torch.utils.data.TensorDataset(perfect_data, perfect_label)
50 r = int(0.95*len(dataset))#何割をにするか管理 val
51 train_dataset, val_dataset = torch.utils.data.random_split(dataset, [r
       , len(dataset)-r])
52 """
53
54 """
55 #に打ち込んでやるぜ Dataloder
56 #batch_size = 30
57 train_loader = DataLoader(train_dataset, batch_size=batch_size,
       shuffle=True)
58 val_loader = DataLoader(val_dataset, batch_size=len(val_dataset),
       shuffle=False)
59
60 print("dataset␣is␣ok")
61 #モデルの定義
62 class Simplemodel(nn.Module):
63     def __init__(self):
64         super().__init__()
65         self.fc1 = nn.Linear(563, 1024)#入力は次元 563
66         self.fc2 = nn.Linear(1024,1024)
67         self.fc3 = nn.Linear(1024, 26)#ラベルの種類は個 26
68         self.criterion = nn.BCEWithLogitsLoss()#損失関数の定義
69         self.optimizer = optim.Adam(self.parameters(), lr=0.0001)
70
71     def forward(self, x):
72         x = self.fc1(x)
73         x = F.relu(x)
74         x = self.fc2(x)
75         x = F.relu(x)
76         x = self.fc3(x)
77         return x
```

```
78
79 def train(model, data_loader):
80     # 今は学習時であることを明示するコード
81     model.train()
82
83     # 正しい予測数，全体のデータ数を数えるカウンターの初期化 0
84     total_correct = 0
85     total_data_len = 0
86
87     loss_mean = 0
88
89     # ミニバッチごとにループさせる,の中身を出し切ったらエポックとなる
           train_loader1
90     for j, (x, t) in enumerate(data_loader):
91         y = model(x) # 順伝播
92         model.optimizer.zero_grad() # 勾配を初期化（前回のループ時の勾配
               を削除）
93         loss = model.criterion(y, t) # 損失を計算
94         loss.backward() # 逆伝播で勾配を計算
95         model.optimizer.step() # 最適化
96
97         loss_mean += loss.item()
98
99         # ミニバッチごとの正答率と損失を求める
100        for i in range(len(t)): # データ一つずつループミニバッチの中身
               出しきるまで,
101            total_data_len += 1 # 全データ数を集計
102            #if index_y[i] == index_t[i]:
103            # total_correct += 1 # 正解のデータ数を集計
104            count = torch.sum(t[i] == 1).item()#正解のやつにいくつが
                   入っているか数える 1
105            top_N_values, top_N_indices_kaitou = torch.topk(t[i],
                   count)## 上位個の要素を取得
                   N)
106            top_N_indices_kaitou_list = top_N_indices_kaitou.tolist()#
                   上位{M個の値を持つ要素のインデックすをリスト化する
                   }
107            #print( "top_N_indices_kaitou_list = ",
                   top_N_indices_kaitou_list)
108            #出力にも同じことをする
109            top_N_values, top_N_indices_syuturyoku = torch.topk(y[i],
                   count)## 上位個の要素を取得
                   N)
110            top_N_indices_syuturyoku_list = top_N_indices_syuturyoku.
```

```
                tolist()#上位{M個の値を持つ要素のインデックすをリスト化する}
111             #print("top_N_indices_syuturyoku_list =", top_N_indices_syuturyoku_list)
112             kyoutu = set(top_N_indices_kaitou_list) & set(top_N_indices_syuturyoku_list)#共通するやつを抜き出す
113
114             total_correct = total_correct + len(kyoutu)/len(top_N_indices_kaitou_list)#一致したやつを全体で割る
115
116     loss_mean = loss_mean / (j+1)
117
118     return total_correct, total_data_len, loss_mean
119
120 def test(model, data_loader):
121     # モデルを評価モードにする
122     model.eval()
123
124     # 正しい予測数，全体のデータ数を数えるカウンターの初期化 0
125     total_data_len = 0
126     total_correct = 0
127
128     loss_mean = 0
129
130     for j, (x, t) in enumerate(data_loader):
131         y = model(x) # 順伝播（予測）=
132         loss = model.criterion(y, t) # 損失を計算
133         loss_mean += loss.item()
134
135         # ミニバッチごとの正答率と損失を求める
136         #print(t,"=t ", y ,"=Y")
137         #_, index_y = torch.max(y, axis=1) # 最も確率が高いと予測した index
138         #_, index_t = torch.max(t, axis=1) # 正解の index
139
140         for i in range(len(t)): # データ一つずつループミニバッチの中身出しきるまで，
141             total_data_len += 1 # 全データ数を集計
142             #if index_y[i] == index_t[i]:
143             # total_correct += 1 # 正解のデータ数を集計
144             count = torch.sum(t[i] == 1).item()#正解のやつにいくつが
```

```
                入っているか数える 1
145             top_N_values, top_N_indices_kaitou = torch.topk(t[i],
                    count)## 上位個の要素を取得
                    N)
146             top_N_indices_kaitou_list = top_N_indices_kaitou.tolist()#
                    上位{M個の値を持つ要素のインデックすをリスト化する
                    }
147             #print( "top_N_indices_kaitou_list = ",
                    top_N_indices_kaitou_list)
148             #出力にも同じことをする
149             top_N_values, top_N_indices_syuturyoku = torch.topk(y[i],
                    count)## 上位個の要素を取得
                    N)
150             top_N_indices_syuturyoku_list = top_N_indices_syuturyoku.
                    tolist()#上位{M個の値を持つ要素のインデックすをリスト化
                    する}
151             #print("top_N_indices_syuturyoku_list =",
                    top_N_indices_syuturyoku_list)
152             kyoutu = set(top_N_indices_kaitou_list) & set(
                    top_N_indices_syuturyoku_list)#共通するやつを抜き
                    出す
153
154             total_correct = total_correct + len(kyoutu)/len(
                    top_N_indices_kaitou_list)#一致したやつを全体で
                    割る
155
156
157     loss_mean = loss_mean / (j+1)
158
159     return total_correct, total_data_len, loss_mean
160
161
162 # アーキテクチャのインスタンス作成
163 model = Simplemodel().to(device)
164
165 epochs = 50
166 record_train_loss = []
167 record_test_loss = []
168 for epoch in range(epochs):
169     train_correct_len, train_data_len, train_loss = train(model,
            train_loader)
170     test_correct_len, test_data_len, test_loss = test(model,
            val_loader)
171
```

```
172     train_acc = train_correct_len/train_data_len*100
173     test_acc = test_correct_len/test_data_len*100
174
175     record_train_loss.append(train_loss)
176     record_test_loss.append(test_loss)
177
178     print(f"epoch={epoch},␣train:{train_correct_len}/{train_data_len}({
            train_acc:.2f}%),␣{train_loss:.5f},␣test:{test_correct_len}/{
            test_data_len}({test_acc:.2f}%),␣{test_loss:.5f}")
179     modelname = 'modelcutbybbox_' + str(epoch)
180     #切り取ったを使う方 clip
181     #PATH = './model_cutbybbox_poseandclip_plus20/'+ modelname
182     #切り取ってない方
183     PATH = './model_cutbybbox/'+ modelname
184     torch.save(model.state_dict(),PATH)#モデルの保存
185     #モデルのロード
186     #model = Simplemodel().to(device)
187     #model.load_state_dict(torch.load(PATH))
```

A.2 で作成したモデルを用いてテストを行うプログラムを A.3 に示す.

ソースコード A.3: test-cutbybbox.py

```
1         import matplotlib
2 import matplotlib.pyplot as plt
3
4 import __init__
5 import vsrl_utils as vu
6 import numpy as np
7
8 import torch
9 import clip
10 import matplotlib.pyplot as plt
11 from PIL import Image
12 import glob
13 import os
14 import numpy as np
15 from torch.utils.data import DataLoader
16 from torch import nn
17 from torch.nn import functional as F
18 from torch import optim
19 import test_withbboxdatareform
```

```
import make_test_cutclip_andpose
from torch.autograd import Variable

import pickle

batch_size = 15 #バッチサイズを管理
coco = vu.load_coco()

#Load the VCOCO annotations for vcoco_train image set
vcoco_all = vu.load_vcoco('vcoco_test')
#画像ロードようのディレクトリ
file_base_dir = os.path.expanduser("~/clip_prefix_caption/
    CLIP_prefix_caption/ClipCap-for-Japanese/val2014/")

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
#切り取ってない方
perfect_data = test_withbboxdatareform.data
perfect_label = test_withbboxdatareform.label

#切り取ったほう
#perfect_data = make_test_cutclip_andpose.data
#perfect_label = make_test_cutclip_andpose.label

#を訓練とに分ける datasetval
dataset = torch.utils.data.TensorDataset(perfect_data, perfect_label)
test_loader = DataLoader(dataset, batch_size=len(dataset), shuffle=
    False)

print("dataset␣is␣ok")
#モデルの定義
class Simplemodel(nn.Module):
    def __init__(self):
        super().__init__()
        self.fc1 = nn.Linear(563, 1024)#入力は次元 563
        self.fc2 = nn.Linear(1024,1024)
        self.fc3 = nn.Linear(1024, 26)#ラベルの種類は個 26
        self.criterion = nn.BCEWithLogitsLoss()#損失関数の定義
        self.optimizer = optim.Adam(self.parameters(), lr=0.0001)

    def forward(self, x):
        x = self.fc1(x)
```

```
59          x = F.relu(x)
60          x = self.fc2(x)
61          x = F.relu(x)
62          x = self.fc3(x)
63          return x
64
65
66  def test(model, data_loader):
67      # モデルを評価モードにする
68      model.eval()
69
70      # 正しい予測数、全体のデータ数を数えるカウンターの初期化 0
71      total_data_len = 0
72      total_correct = 0
73
74      loss_mean = 0
75
76      for j, (x, t) in enumerate(data_loader):
77          y = model(x) # 順伝播（予測）=
78          loss = model.criterion(y, t) # 損失を計算
79          loss_mean += loss.item()
80
81          # ミニバッチごとの正答率と損失を求める
82          #print(t,"=t ", y ,"=Y")
83          #_, index_y = torch.max(y, axis=1) # 最も確率が高いと予測した
                index
84          #_, index_t = torch.max(t, axis=1) # 正解の index
85
86          for i in range(len(t)): # データーつずつループミニバッチの中身
                出しきるまで,
87              total_data_len += 1 # 全データ数を集計
88              #if index_y[i] == index_t[i]:
89              # total_correct += 1 # 正解のデータ数を集計
90              count = torch.sum(t[i] == 1).item()#正解のやつにいくつが
                    入っているか数える 1
91              top_N_values, top_N_indices_kaitou = torch.topk(t[i],
                    count)## 上位個の要素を取得
                    N)
92              top_N_indices_kaitou_list = top_N_indices_kaitou.tolist()#
                    上位{M個の値を持つ要素のインデックすをリスト化する
                    }
93              #print( "top_N_indices_kaitou_list = ",
                    top_N_indices_kaitou_list)
```

```
94              #出力にも同じことをする
95              top_N_values, top_N_indices_syuturyoku = torch.topk(y[i],
                    count)## 上位個の要素を取得
                    N)
96              top_N_indices_syuturyoku_list = top_N_indices_syuturyoku.
                    tolist()#上位{N個の値を持つ要素のインデックすをリスト化
                    する}
97              #print("top_N_indices_syuturyoku_list =",
                    top_N_indices_syuturyoku_list)
98              kyoutu = set(top_N_indices_kaitou_list) & set(
                    top_N_indices_syuturyoku_list)#共通するやつを抜き
                    出す
99
100             total_correct = total_correct + len(kyoutu)/len(
                    top_N_indices_kaitou_list)#一致したやつを全体で
                    割る
101
102
103     loss_mean = loss_mean / (j+1)
104
105     return total_correct, total_data_len, loss_mean
106
107
108 def test2(model, data_loader):
109     # モデルを評価モードにする
110     model.eval()
111
112     # 正しい予測数、全体のデータ数を数えるカウンターの初期化 0
113     total_data_len = 0
114     total_correct = 0
115
116     loss_mean = 0
117
118     for j, (x, t) in enumerate(data_loader):
119         y = model(x) # 順伝播（予測）=
120         loss = model.criterion(y, t) # 損失を計算
121         loss_mean += loss.item()
122
123         # ミニバッチごとの正答率と損失を求める
124         #print(t,"=t ", y ,"=Y")
125         #_, index_y = torch.max(y, axis=1) # 最も確率が高いと予測した
                index
126         #_, index_t = torch.max(t, axis=1) # 正解の index
```

```
127
128         for i in range(len(t)): # データ一つずつループミニバッチの中身出しきるまで,
129             total_data_len += 1 # 全データ数を集計
130             #if index_y[i] == index_t[i]:
131             # total_correct += 1 # 正解のデータ数を集計
132             count = torch.sum(t[i] == 1).item()#正解のやつにいくつが入っているか数える 1
133             top_N_values, top_N_indices_kaitou = torch.topk(t[i], count)## 上位個の要素を取得N)
134             top_N_indices_kaitou_list = top_N_indices_kaitou.tolist()#上位{N個の値を持つ要素のインデックすをリスト化する}
135             #print( "top_N_indices_kaitou_list = ", top_N_indices_kaitou_list)
136             #出力にも同じことをする
137             top_N_values, top_N_indices_syuturyoku = torch.topk(y[i], count)## 上位個の要素を取得N)
138             top_N_indices_syuturyoku_list = top_N_indices_syuturyoku.tolist()#上位{N個の値を持つ要素のインデックすをリスト化する}
139             #print("top_N_indices_syuturyoku_list =", top_N_indices_syuturyoku_list)
140             kyoutu = set(top_N_indices_kaitou_list) & set(top_N_indices_syuturyoku_list)#共通するやつを抜き出す
141             if len(kyoutu) ==count:
142                 total_correct += 1#一致したやつを全体で割る
143
144
145     loss_mean = loss_mean / (j+1)
146
147     return total_correct, total_data_len, loss_mean
148
149
150
151 i = 0
152 model = Simplemodel().to(device)
153 while i < 50:
154     modelname = 'modelcutbybbox_' + str(i)
155     #切り取ったほう
156     #PATH = './model_cutbybbox_poseandclip_plus20/'+ modelname
```

```
157     #切り取ってない方
158     PATH = './model_cutbybbox/'+ modelname
159     model.load_state_dict(torch.load(PATH))
160     test_correct_len, test_data_len, test_loss = test2(model,
            test_loader)
161
162     record_test_loss = []
163     test_acc = test_correct_len/test_data_len*100
164     record_test_loss.append(test_loss)
165     print(f"i:{i},test:{test_correct_len}/{test_data_len}({test_acc:.2f
            }%),␣{test_loss:.5f}")
166     i +=1
```