

卒業研究報告題目

Swin Transformerの画像分類性能に対する
パッチサイズの影響

Effect of Patch Size on Swin Transformer's Image
Classification Performance

指導教員 出口 利憲 教授

岐阜工業高等専門学校 電気情報工学科

2020E01 赤谷 隼

令和 7 年（2025 年） 2 月 17 日提出

Abstract

AI has made remarkable progress in recent years, and a model called Swin Transformer has attracted considerable attention in image classification tasks.

The Swin Transformer has a computational method that captures image features by dividing the image into patches; in Swin Transformer, the default patch size is 4x4. The purpose of this study is to vary this patch size from 2x2 to 8x8 to train for image classification using 10 classes from the dataset “Food101” to find a patch size that can classify efficiently in terms of classification accuracy and execution time.

The results showed 5%~10% higher classification at patch sizes 4 and 6 than at other patch sizes. Patch size 6 was also found to be more efficient than at patch size 4 for calculating about 2 hours earlier with the same accuracy.

The problems with this experiment are that we limited the input image size to 224 x 224, and we did not use a trained model that is likely to be used frequently in practical applications. Therefore, we would like to study different image sizes and other models.

目次

Abstract	i
第1章 序論	1
第2章 ニューラルネットワーク	2
2.1 ニューロン	2
2.2 ニューラルネットワーク	2
2.3 勾配降下法	3
2.4 活性化関数	5
2.5 損失関数	7
2.6 過学習	8
第3章 Swin Transformer	13
3.1 Transformer	13
3.2 Self-Attention	13
3.3 ViT	14
3.4 Swin Transformer	15
3.4.1 Patch Partition と Linear Embedding	17
3.4.2 Patch Merging	17
3.4.3 Swin Transformer Block	17
第4章 実験	20
4.1 実験目的	20
4.2 実験方法	20
4.3 評価方法	21
4.4 環境設定	21
4.5 モデル詳細	22
4.6 Data Augmentation	23
4.7 実験結果 (分類精度)	23
4.8 実験結果 (計算速度)	28
4.9 考察	28
第5章 結論	31

参考文献	32
謝辞	33

第1章 序論

近年、人工知能はめざましい進化を遂げている。様々な人工知能が研究されている中で、画像分類の分野への需要も高まってきている。私の住む雄志寮で現在顔認証システムを利用して点呼を行っているように、様々な場面での活用が期待されている。

今後セキュリティシステムとして顔認証を用いることや製品の仕分けをより早く正確に判断する必要があることを考慮すると、画像分類モデルの精度と処理速度のさらなる向上は、研究者にとって止め処ない課題である。

本研究で用いる Swin Transformer¹⁾ は、Ze Liu 氏らにより 2021 年に新たに提案され、同年の画像処理分野の学会である International Conference on Computer Vision (ICCV) においてベストペーパーを受賞したものである。当時画像分野に応用され始めた Transformer を基盤にした革新的なモデルであり、今後の活用が期待されるモデルである。

本研究では、Swin Transformer の特徴の一つである、画像をパッチに分割してパッチごとに計算を行うことで画像の特徴を捉える計算手法に注目する。このパッチのサイズを変更して学習をすることで、分類精度や計算速度にどのような影響を及ぼすのかを調査していく。

第2章 ニューラルネットワーク

2.1 ニューロン

実際のヒトの脳神経ネットワークの構成要素としてのニューロンの模式図を Figure 2.1 に示す。実際の神経細胞の形態はより多様だが、ニューロンとは基本的には細胞体から樹状突起と軸索がのびたものである。軸索の先は枝分かれして他のニューロンの樹状突起や細胞体と結合部(シナプス)をつくる。電気的情報は細胞体から軸索、シナプスを経て次の細胞に伝わる。ヒトの脳の中には約 1.4×10^{10} 個のニューロンが存在する。1 個のニューロンが $10^3 \sim 10^4$ 個のシナプスを形成しており、脳全体としては $10^{13} \sim 10^{14}$ 個のシナプスが存在すると推定されている。²⁾

このようなヒトの脳内のニューロンを、簡略化しモデル化したものをニューロンモデルと呼ぶ。ニューロンモデルを示した図を Figure 2.2 に示す。1 つのニューロンには複数の入力が存在し、それぞれに重みをかけ合わせて総和を取る。そしてこの値に活性化関数 f により何らかの処理を加えた値が出力となる。³⁾

2.2 ニューラルネットワーク³⁾

ニューラルネットワークとは、人間の脳の仕組みを模倣し、従来のアルゴリズムでは困難だった問題を推論できる機械学習モデルのことである。Figure 2.3 のように、ニューラルネットワークは入力層、中間層(隠れ層)、出力層の3つに分類される層で構成されている。

入力層は、ニューラルネットワーク全体の入力を受け取り、出力層はネットワーク全体の出力を出力する。中間層は入力層と出力層の間にある複数の層である。これらのうち、ニューロンの演算が行われるのは中間層と出力層のみであり、入力層は受け取った入力を中間層に渡すのみである。通常のニューラルネットワークでは、1 つのニューロンからの出力が、次の層のすべてのニューロンの入力とつながっている。基本的に、層の数が多くなるほどネットワークの表現力は向上するが、それに伴って学習が難しくなっていく。

ニューラルネットワーク全体の出力と、正解の誤差がより小さくなるように重みやバイアスなどのパラメータを調整していくことで、ネットワークは次第に学習し、適切な予測をすることができるようになる。このとき、一般的には出力層から1層ずつ遡るよう

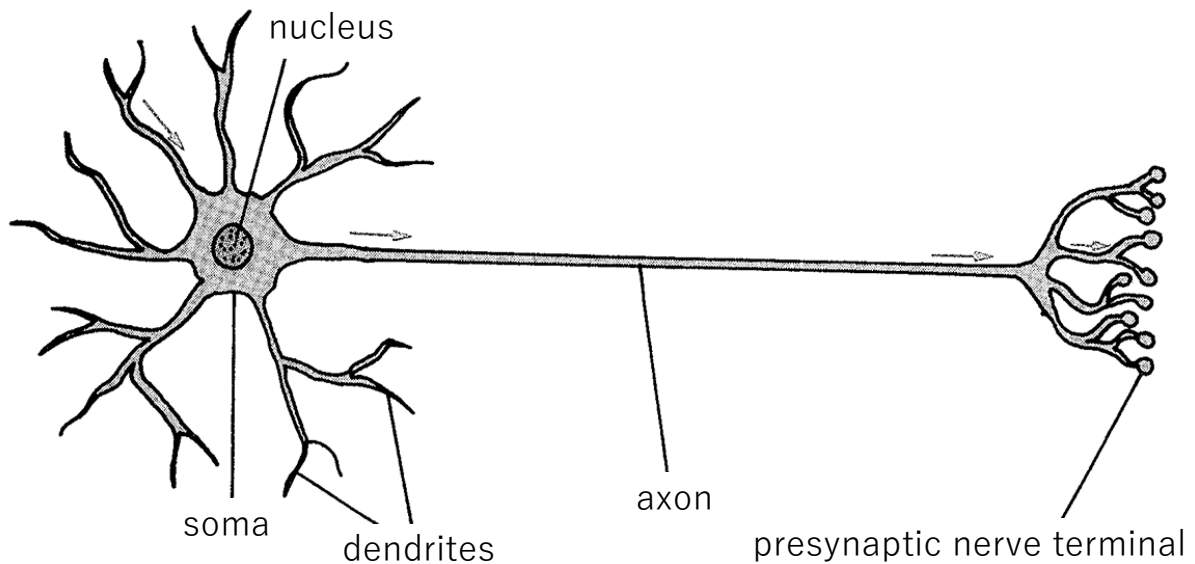


Figure 2.1 Schematic diagram of neuron.²⁾

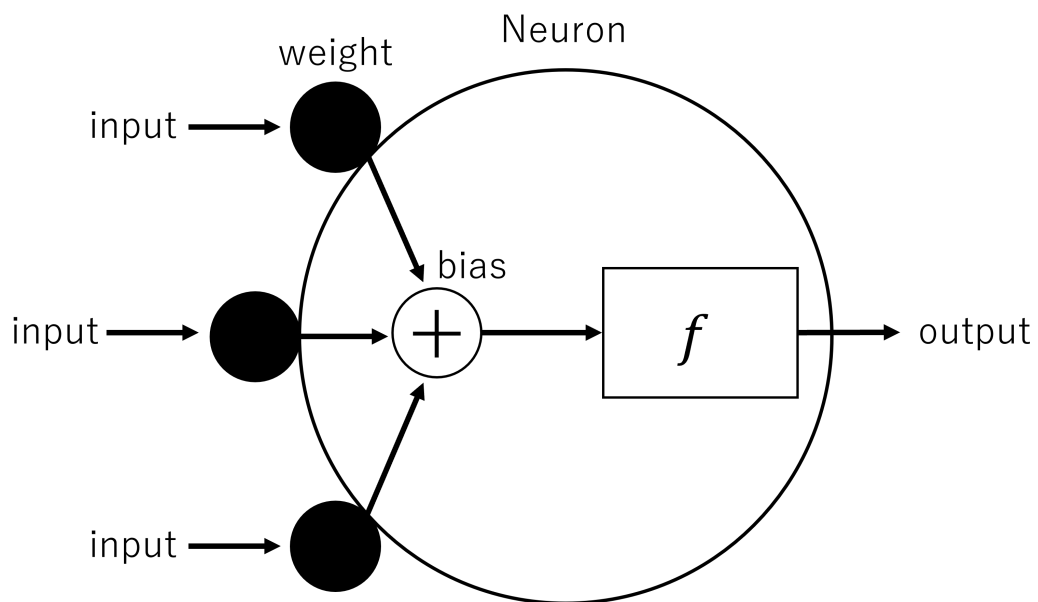


Figure 2.2 Neuron model.

にして誤差を伝播させ、これを基に重みとバイアスを更新する手法が用いられる。このアルゴリズムをバックプロパゲーション、もしくは誤差逆伝播法と呼ばれる。Figure 2.3 にバックプロパゲーションの概要を示す。

2.3 勾配降下法³⁾

バックプロパゲーションでは、勾配降下法によってパラメータの修正量が決定される。勾配降下法のイメージを、Figure 2.4 に示す。

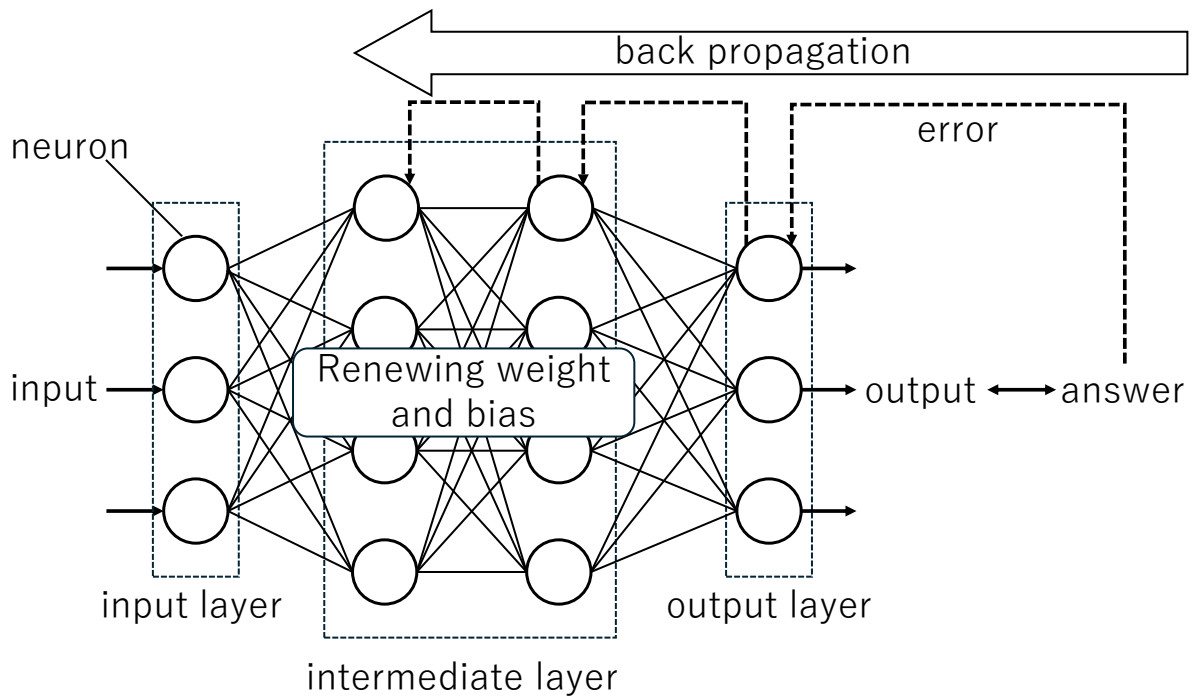


Figure 2.3 Neural network and back propagation.

このグラフでは、横軸の w がある重み、縦軸の E が誤差を示す。 $\frac{\partial E}{\partial w}$ は誤差 E を重み w で偏微分したもので、これは曲線の傾き (勾配) を表す。重みの値に応じて誤差は変化するが、実際はこのような曲線の形状を知ることはできないので、足元の曲線の傾きに応じて少しずつ重みを変化させていく。ネットワークのすべての重みをこの曲線を降下するように変化させていけば、誤差を次第に小さくしていくことができる。したがって、ニューラルネットワークのすべての重みとバイアスを更新するためにまず必要なことは、すべての重みとバイアスに対する誤差の勾配を求めることである。

重みとバイアスの更新は、最も簡単な確率的勾配降下法 (Stochastic Gradient Descent, SGD) の場合、偏微分を用いた次の式 (2.1) で表される。

$$w \leftarrow w - \eta \frac{\partial E}{\partial w} \quad (2.1)$$

$$b \leftarrow b - \eta \frac{\partial E}{\partial b} \quad (2.2)$$

ここで w は重み、 b はバイアス、 E は誤差で矢印はパラメータの更新を表す。また、 η は学習係数と呼ばれる定数で、学習の速度を決める係数である。ニューラルネットワークにおけるすべての勾配を求めたあと、上記の式に基づいてすべてのニューロンの重みとバイアスを更新し、次の学習に進むことになる。

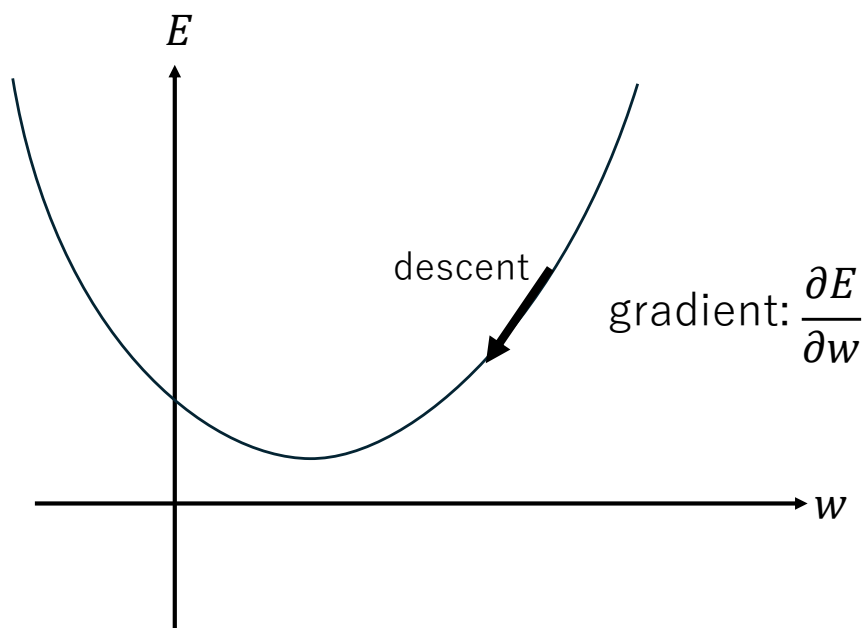


Figure 2.4 Gradient descent method.

2.4 活性化関数⁴⁾

2.1 節で述べた通り、活性化関数とは、ニューロン内の入力に重みを乗算しバイアスを加えた総和から、出力を決定する関数のことである。ニューラルネットワークでは様々な活性化関数が用いられてきたが、近年では ReLU 関数や、分類問題ではシグモイド関数や softmax 関数が主流である。

1. シグモイド関数

Figure 2.5 は、式 (2.3) で表されるシグモイド関数である。シグモイド関数は、機械学習の 2 クラス識別モデルの出力層としてよく用いられる。ニューラルネットワークの出力ベクトルを、(0,1) の範囲に押しつぶしてベルヌーイ分布として出力できる。微分の計算が容易に行えるため、勾配降下法に向いているという特徴を持つ。

$$f(x) = \frac{1}{1 + \exp(-x)} \quad (2.3)$$

2. ReLU 関数

Figure 2.6 は、式 (2.4) で表される Rectified Linear Unit(ReLU) 関数であり、コンピュータビジョンでの認識タスクなどのニューラルネットワークの隠れ層に用いられる活性化関数である。正の範囲の入力そのまま出力し、負の範囲の入力を 0 にして出力する。従来のシグモイド関数などで発生する勾配消失が起きず、正

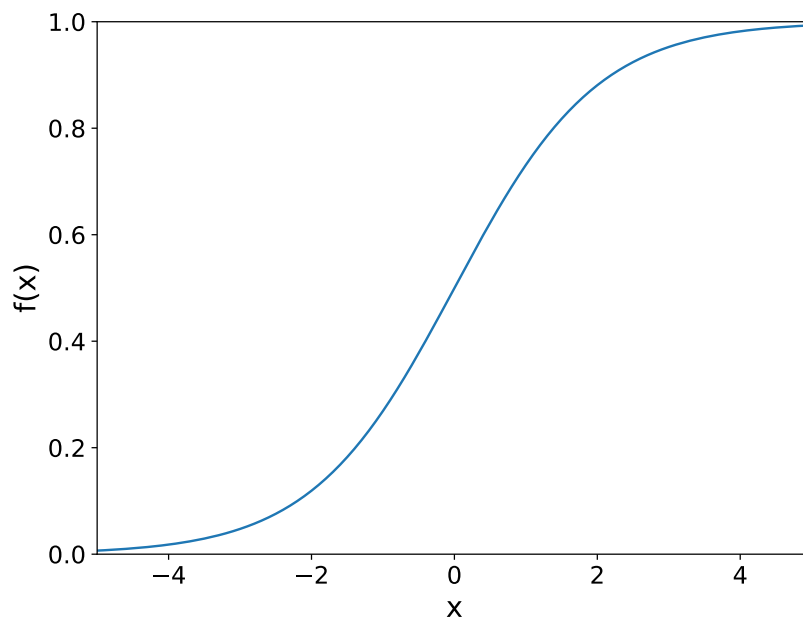


Figure 2.5 Sigmoid function.

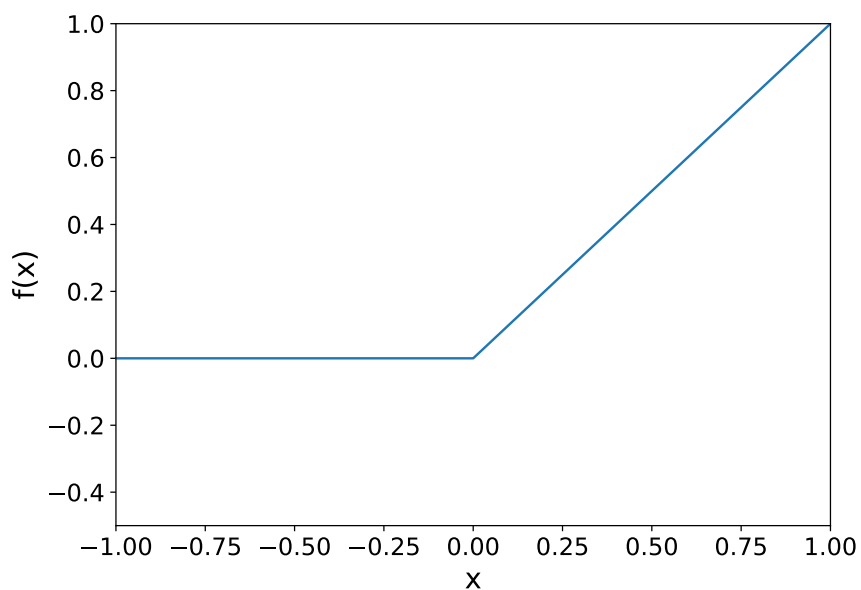


Figure 2.6 ReLU function.

の範囲で微分値が常に 1 であることから、安定性が高く、学習の所要時間が短くなるのが特徴である。

$$f(x) = \max(0, x) \quad (2.4)$$

3. softmax 関数

softmax 関数は、 K クラス識別モデルの出力の、クラス i の確率をそれぞれ独立に計算する関数である。定義式を式 (2.5) に示す。機械学習の多クラス識別モデ

ルにおいて、出力層に用いられる関数である。合計が1になるようにベクトル中の各次元の変数を正規化する関数であり、シグモイド関数の多変数向けの一般化である。

$$f(x)_i = \frac{\exp(x_i)}{\sum_{k=1}^K \exp(x_k)} \quad (2.5)$$

2.5 損失関数⁵⁾

機械学習における損失関数 (Loss Function) とは、機械学習モデルが予測した値と実際の正解値との間の差異を定量的に評価するための関数である。モデルの予測の誤差を測る指標となり、損失関数の出力値が小さいほど、モデルの予測が正確であることを意味する。

画像入力の $C(\geq 2)$ クラス分類問題を学習する際、損失関数には、交差エントロピー誤差をサンプル数だけ和を取った交差エントロピー損失が標準的に用いられる。ここで、交差エントロピー (Cross Entropy: CE) は、情報学の定義では、離散確率分布 p と q の距離測度として、式 (2.6) のように定義される。

$$H(p, q) = - \sum_{x \in \mathcal{X}} p(x) \log q(x) \quad (2.6)$$

交差エントロピー $H(p, q)$ は、 p と q の間で取ったエントロピー (平均情報量) である。そして、これは p と q の2分布が離れている度合いの尺度を表す。 p と q が同じ確率分布であるほど交差エントロピー値はゼロに近づくので、予測値が確率の値であるときの機械学習モデルの誤差関数として用いることが多い。

式 (2.7) の交差エントロピー損失は、データ数を N 、クラス数を K としたときの、ターゲットのクラス確率ベクトル $\mathbf{y}$ に、出力のクラス確率ベクトル $\hat{\mathbf{y}}_k$ をフィットさせる関数である。

$$\mathcal{L}_{\text{CE}} = - \sum_{i=1}^N \sum_{k=1}^K y_k^{(i)} \log \hat{y}_k^{(i)} \quad (2.7)$$

交差エントロピー損失は、各次元 k における負の対数尤度の和で計算されることから、負の対数尤度 (Negative log likelihood) 損失とも呼ばれる。

多クラス分類問題を学習する際、交差エントロピー損失+softmax 出力が基本的な組み合わせとなる。各ニューロン間の個別重みを w_{ij} 及び w_{jk} とし、 w_{jk} から w_{ij} へ逆伝播するとすると、式 (2.8) のように計算コストが低い微分値を w_{ij} へ逆伝播することができる。

$$\frac{\partial \mathcal{L}}{\partial w_{jk}} = \frac{y_k - \hat{y}_k}{\hat{y}_k(1 - \hat{y}_k)} \hat{y}_k(1 - \hat{y}_k)x_j \quad (2.8)$$

$$= (y_k - \hat{y}_k)x_j \quad (2.9)$$

ここで、式 (2.8) の、 $\frac{y_k - \hat{y}_k}{\hat{y}_k(1 - \hat{y}_k)}$ は交差エントロピーの微分関数、 $\hat{y}_k(1 - \hat{y}_k)$ は softmax 関数の微分関数である。

主要な Deep Learning ライブラリでも、計算が高速になるのでこの組み合わせが実用されている。例えば PyTorch では、交差エントロピーと softmax 関数を、個別にそれぞれ計算するのではなく、合成して計算コストを減らした式 (2.8) で逆伝播できる `torch.nn.CrossEntropyLoss` クラスが実装されている。

また、実用上では、softmax 関数のあとに対数関数も追加した「softmax -> log」構成の出力層にすることも多い。softmax の対数関数のおかげで、特定のクラスの生起確率が小さくなりすぎてアンダーフローが生じ、その計算誤差によって学習やテストが止まってしまうことを防ぐことができるためである。前述の `CrossEntropyLoss` クラスの中には、この「softmax -> log」構成が使用されている。

2.6 過学習^{6), 7)}

過学習とは、Figure 2.7 のように機械学習モデルがトレーニングデータに対しては正確な予測をするが、トレーニングデータをコンピュータが学習しすぎた結果、過剰に適合しすぎてしまったことで評価用の新しいデータについては正確に予測しないという、望ましくない機械学習の動作である。「オーバーフィッティング (Overfitting)」や「過剰適合」とも呼ばれる。あらかじめ用意されたデータの正解率が高くても、実際の運用で使うテストデータにおいて精度が出ないのであれば役に立たないモデルとなってしまうた

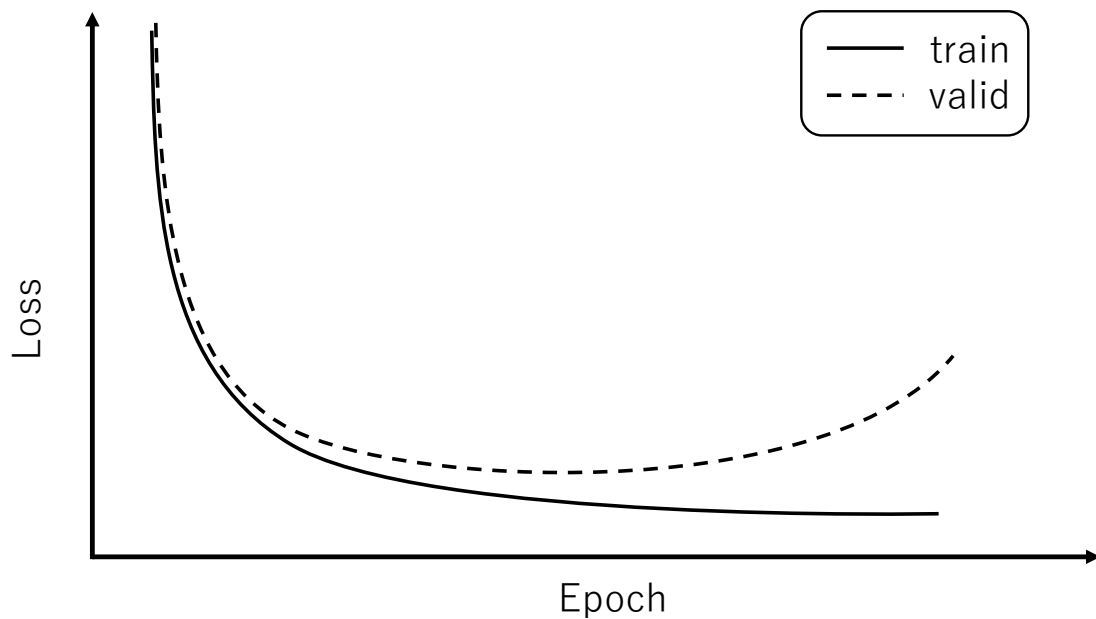


Figure 2.7 Overfitting.

め、過学習を起こさず幅広いデータのインプットに対して正しい推測ができることが機械学習では重要である。

過学習が発生するのは、次のような原因がある。

- トレーニングデータのサイズが小さすぎて、考えられるすべての入力を正確に予測するのに十分なデータサンプルが不足している。
- 無関係な情報(ノイズを含むデータ)がトレーニングデータに大量に含まれている。
- 1つのサンプルデータセットに対して、モデルのトレーニング時間が長すぎる。
- モデルの複雑度が高いことが原因で、トレーニングデータ内のノイズを学習してしまう。

過学習が発生してしまった場合、次のような手法をとることで抑制することができる。

1. 正則化

正則化 (Min-Max Normalization) とは、複雑化してしまったモデルを単純なモデルへと戻していく手法である。特徴量を重要度に基づいて等級付けすることで、予測結果に影響を与えない要因の排除を試みる。正則化には、主に「L1 正則化」と「L2 正則化」がある。L1 正則化は、データ数や特徴量が多い場合に余分な特徴量を減らすことで過学習を防ぐ手法で、本当に必要な変数だけをモデルに利用したいときに役に立つ。L2 正則化は、重みが大きい特徴量の値を下げて1つの変数の影響を下げることで過学習を防ぐ手法で、L1 正則化を使ったモデルと比較す

るとより予測精度が高くなる傾向がある。

2. ドロップアウト

ドロップアウト (Dropout) とは、学習中の各エポックにおいて、各層でベルヌーイ分布などに基づいてランダムに特定のニューロン間の重みを 0 にして無効化することにより、極めて簡単な実装かつ低い計算負荷で強力な正則化を行うことができる手法である。モデルは解決したい問題の複雑さに合わせて作られているため前述の正則化のような変更が簡単にできないことがほとんどだが、ドロップアウトを行うことでモデルの簡素化と同様の効果を得ることができる。ドロップアウト層は、全結合層・畳み込み層直後の各 ReLU 活性化関数の次に配置することが標準的なネットワーク設計である。

ドロップアウトのイメージ図を Figure 2.8 に示す。学習中に各エポックごとに、全体のうち割合 p のニューロンだけをランダムに無効化し、残ったユニットのみで構成されるサブネットワークで順伝播と逆伝播による重みの更新を行う。ドロップ率 p は、通常 0.1~0.5 程度に設定するが、学習データ量とモデル規模のバランス次第で、どの程度の強さでドロップアウトするかをユーザーが見積もって決める必要がある。重みの更新では、エポックごとに残されたニューロンの重みのみをそのエポックで更新する。そして、毎回のエポックでは異なるユニット間をドロップして他のエポックとは異なるサブネットワークを形成し、順伝播・損失の計算と重みの逆伝播を行う。これにより各ニューロンはよりロバストになる上、間引かれた伝播になるため学習が速くなる効果を得られる。テスト時には、学習時の間引きされた層で計算していた状態との整合性を保つため、学習時にドロップした各ユニットから次に接続する重み w_i を、 p 倍した pw_i に差し替える。

3. Data Augmentation

Data Augmentation(データ拡張)とは、学習用の画像データに対して変換を施すことでデータを水増しする方法である。Data Augmentation の適用例を Figure 2.9 に示す。図のように、元ある画像に対して回転や平行移動、拡大縮小、反転、モデルによっては色調の変更などを行う。

近年の画像認識モデルには非常に深い層を持つ物が多く、そういったモデルは

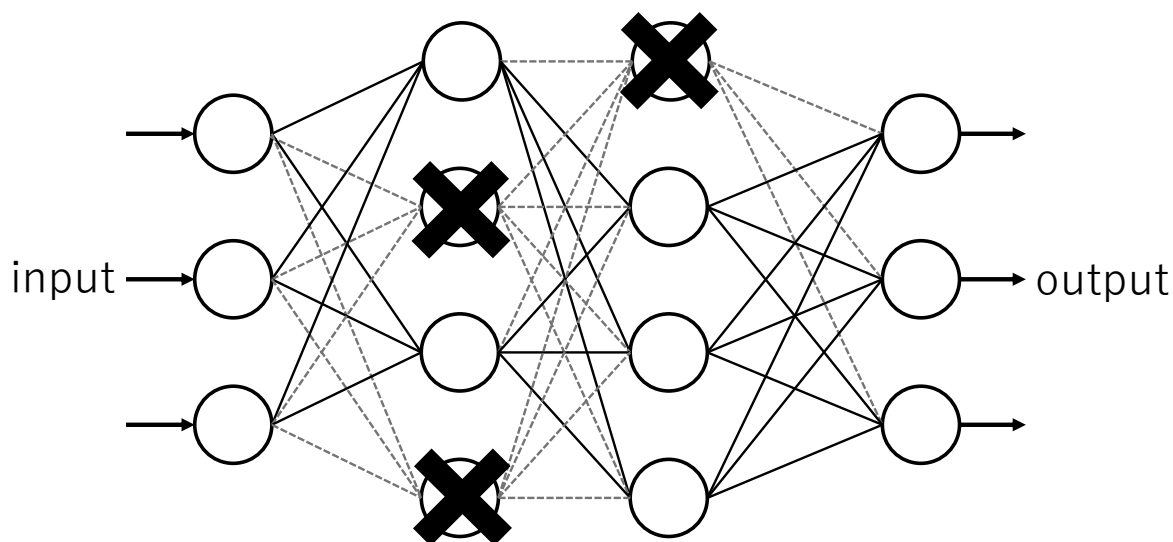


Figure 2.8 Dropout.

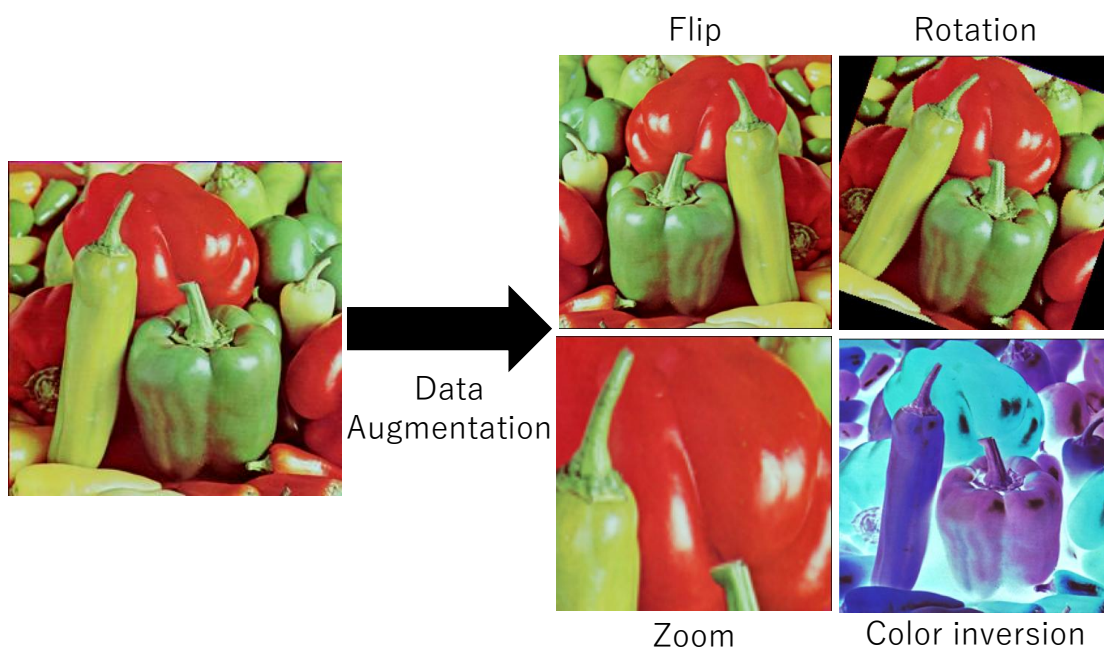


Figure 2.9 Data Augmentation.

多くのパラメータを有するため、学習には大量のデータが必要とされる場合がある。しかし、データによっては必要な数を用意できない場合がある。そこで、Data Augmentation により、少ないデータ数を水増しすることでより多くの画像を学習に用いることができるようになる。

また、データ数が少ない場合や似たような画像を多く入力してしまった場合において、画像認識モデルは過学習を起こしてしまいやすいため、Data Augmentation

を用いてバリエーションに富んだ画像を入力することで、よりモデルの汎化性能が高まる。例えば、犬と猫を分類しようとする際に、入力画像のすべての犬が左を向いており、仮にすべての猫が右を向いていた場合、モデルが犬と猫の生物学的な特徴から判断するのではなく体の向きで判断するようになってしまう過学習が起こり得る。このようなデータに対し Data Augmentation を適用し、左右を反転させた画像を加えることで汎化性能を高めることができる。

Data Augmentation は、適用するタイミングによってオフライン拡張とオンライン拡張に分けられる。オフライン拡張はデータセットの画像自体に適用し、単純に画像の枚数を増やす手法である。一方で、オンライン拡張はデータセットを分割したバッチに対して適用する方法である。この方法ではエポックごとにランダムな変換を適用することが可能になり、より多くの画像を学習に使用することができるようになる上に、オフライン拡張とは異なりデータセット自体の容量を大きくしない点で極めて有効である。

第3章 Swin Transformer

3.1 Transformer⁸⁾

2017年に論文“Attention is All You Need”にて提案された新たなモデルが、Transformerである。タイトル中の Attention(注意機構)とは、目的のタスクを解くのに重要な情報を選択する役割を担う。現在では自然言語処理の様々なタスクに Transformer が活用されているが、当時は機械翻訳のための手法として提案されたモデルである。これまで用いられてきた Recurrent Neural Network(RNN, 回帰型ニューラルネットワーク)や Convolutional Neural Network(CNN, 畳み込みニューラルネットワーク)を使わず、機械翻訳タスクにおいて当時の最高性能を更新したことで大きく注目を集めた。RNN や CNN にはできなかった計算の並列化と離れたトークン同士の関係の捕捉を実現するため、Self-Attention 機構を活用している。

3.2 Self-Attention⁸⁾

Transformer で極めて重要な役割を持つのが、Attention(注意機構)である。Attention とは、2つの系列の間で、各要素の関連度合いを計算する手法である。Figure 3.1 に Attention 機構のイメージを示す。画像や単語などのトークン 3 つからなる 2 つの系列に対して Attention 機構を適用した様子を表している。要素同士の関連度合いを線の濃さで表現している。

Transformer や後述する ViT、Swin Transformer などでは Self-Attention(自己注意機構)が提案されている。Self-Attention とは、同じ系列に対して Attention 機構を適用する手法である。つまり、Figure 3.1 における Series1 と Series2 が同じ系列となる。Self-Attention を「AI is very interesting.」に適用したときのイメージ図を Figure 3.2 に示す。ここで、高い関連度を示す線のみを表示している。

関連度をもとにした出力ベクトルの計算方法は次のとおりである。入力されたベクトルを異なる 3 つの線形層でそれぞれ埋め込み、そのあとの各ベクトルをクエリベクトル、キーベクトル、バリューベクトルと呼ぶ。各単語に対応するクエリベクトル列とキーベクトル列に Self-Attention を適用する。例えば「AI」に対応する出力ベクトルは、「AI」ベクトルから見た各キーベクトルとの関連度を反映して計算され、関連度はすべてを足して 1 になるように計算される。この関連度をバリューベクトルに掛けて和をとること

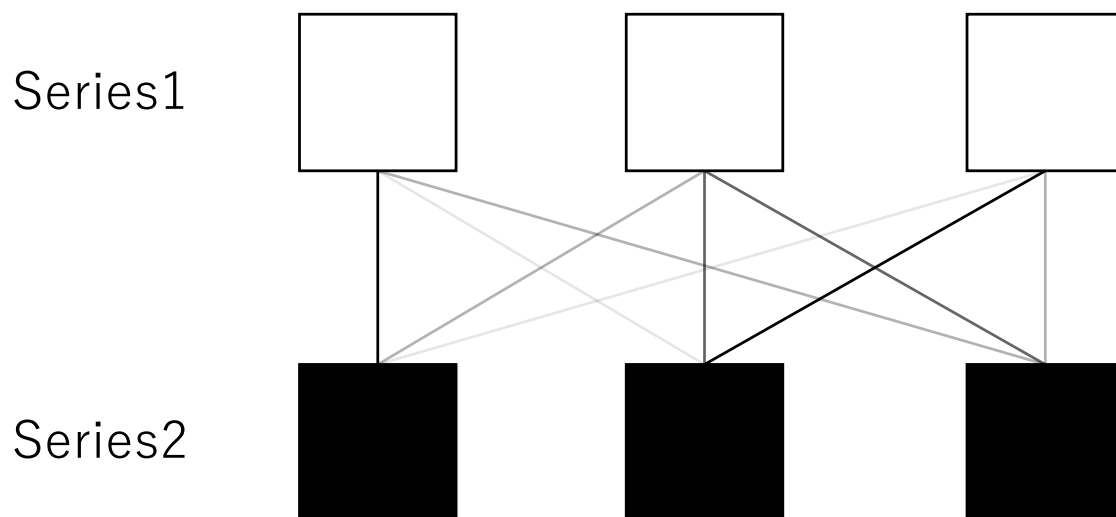


Figure 3.1 Attention

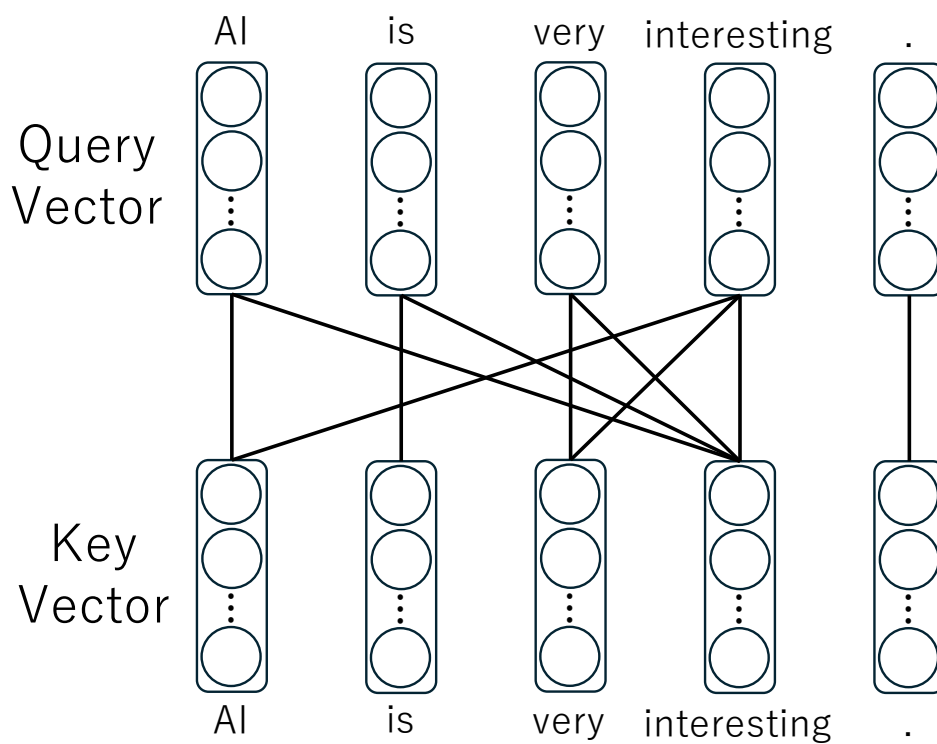


Figure 3.2 Self-Attention.

で出力を得る。

3.3 ViT⁸⁾

2017年にTransformerが登場して以降、自然言語処理分野の様々なベンチマークにおいてBERTやGPTなどのTransformerをもとにしたモデルが高い性能を示している。

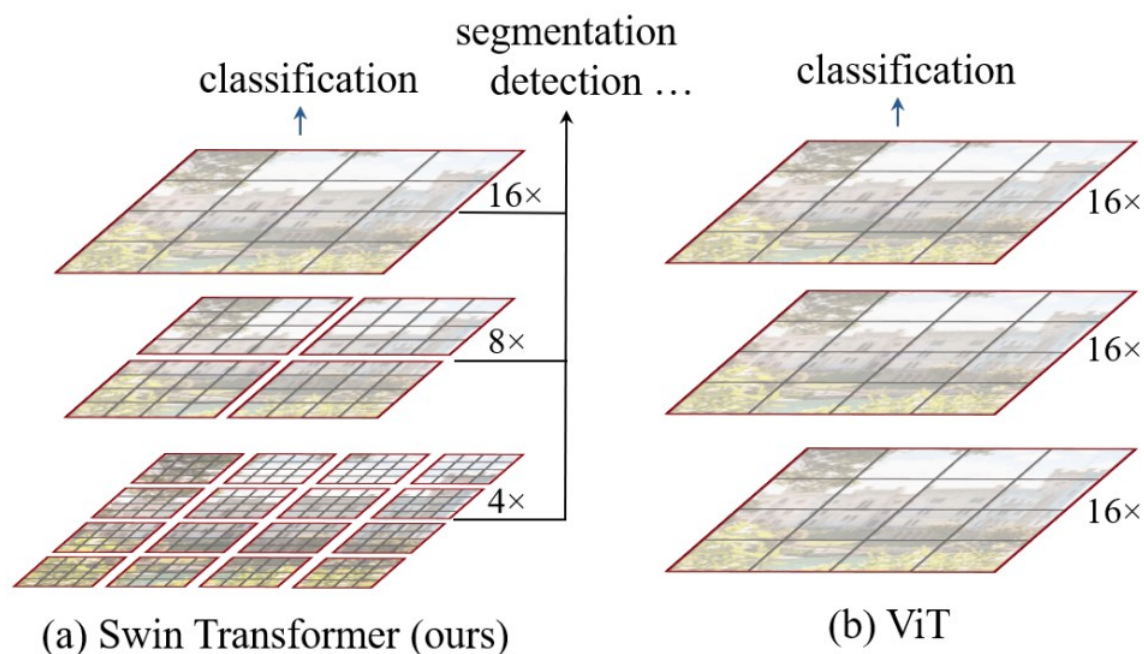


Figure 3.3 Self-Attention computation range of Swin Transformer and ViT.¹⁾

一方で、コンピュータビジョン分野においては、長らく ResNet や EfficientNet をはじめとした CNN ベースのモデルが主流であった。ところが 2020 年 10 月、Transformer ベースのモデルである ViT (Vision Transformer) が提案された。ViT は畳み込みの代わりに Self-Attention を用いていることが特徴である。ViT では、入力画像を均等なサイズのパッチに区切ってトークンとして扱う。パッチ中のピクセル値を並べてベクトルとみなし、これを線形変換したものを各トークンのベクトル表現として扱う。このベクトルに対して Self-Attention を適用することで、画像分類のタスクにおいて SoTA を達成した。ViT の登場以降、これを改良した様々なモデルが数多くのベンチマークで SoTA を更新し続けている。

3.4 Swin Transformer⁸⁾

ViT は高い認識精度を達成する一方で、Self-Attention の計算量が入力の長さ (パッチ数) の二乗に比例する。画像認識では解像度の高い画像を扱うタスクが多く、それらに ViT の構造をそのまま適用するのは困難である。また、Figure 3.3(b) のように、ViT の画像を特定サイズのパッチに分割しているため、多様な物体スケールの変化を捉えることができない。このような ViT の問題点を緩和した手法が、Swin Transformer である。

Swin Transformer では、Figure 3.3(a) のように細かく分割された画像パッチを深い層で

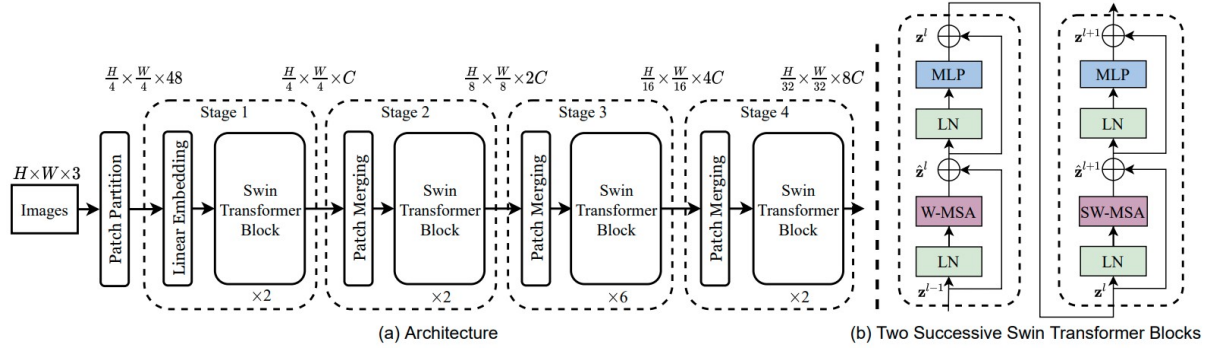


Figure 3.4 Swin Transformer network structure.¹⁾

結合することで階層的な特徴マップを構築する。このとき、広く分割された Window(局所窓) 内のみで Self-Attention を計算する。具体的には、Figure 3.3(a) の最下層の場合、 16×16 に細かく分割したパッチから、 4×4 に広く分割した Window 内のみで Self-Attention を計算する。この構造によって計算量問題や多様な物体スケールに対応することができる。

Swin Transformer のネットワーク構造を Figure 3.4 に示す。ここで、 H 及び W は、それぞれ入力画像の高さと幅を示している。Swin Transformer は Stage ごとに特徴マップのサイズを小さくする構造を持っており、これらの Stage が CNN のような階層的構造を表現している。Swin Transformer は、Figure 3.4(a) で示すように、4つのブロックで画像特徴量を抽出する。まず、Patch Partition で画像を ViT のように重複しないパッチに分割する。Stage1 では、分割したパッチから Linear Embedding でパッチ特徴量を得る。そして Swin Transformer Block でパッチ間の対応関係を取得する。Swin Transformer Block の中身は Figure 3.4(b) に示すように、偶数番目の層で Window based Multi-head Self-Attention(W-MSA)、奇数番目の層で Shifted Window based Multi-head Self-Attention(SW-MSA) を用いてパッチ間の対応関係を捉える。Stage2 では、Patch Merging を用いて特徴マップサイズを小さくする。具体的には、 2×2 の隣接するパッチ特徴量をチャンネル C 次元方向に連結し、 $4C$ 次元に連結された特徴量に対し、全結合層を適用してチャンネルが $2C$ 次元になるよう線形変換する。その後は再び Swin Transformer Block でパッチ間の対応を捉える、といった流れをそれ以降の Stage で重ねていく。最終的に特徴マップのサイズを $\frac{H}{32} \times \frac{W}{32}$ ピクセルまで小さくしたあと、Global Average Pooling と全結合層で分類を行う。

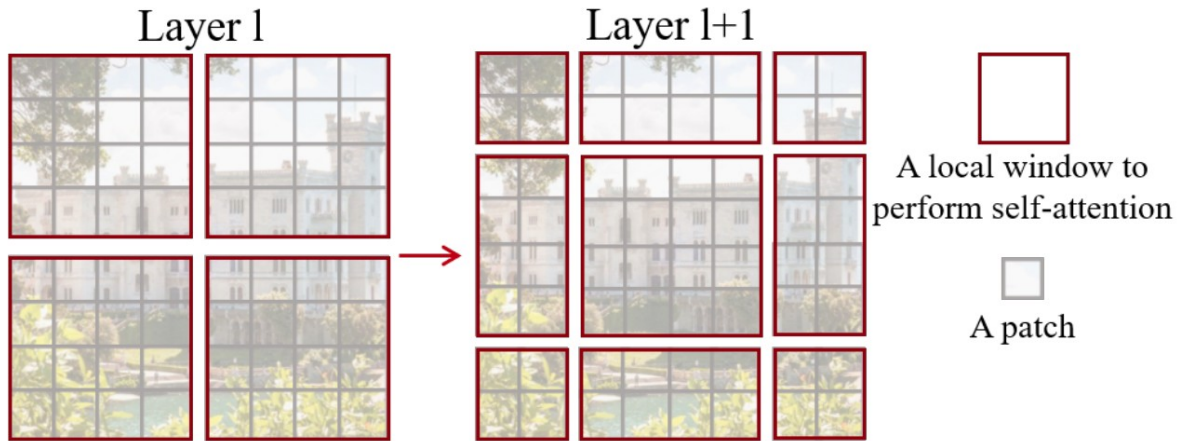


Figure 3.5 Shifted Window.¹⁾

3.4.1 Patch Partition と Linear Embedding

前述の通り、Swin Transformer では Patch Partition で固定サイズのパッチに分解し、Linear Embedding で各パッチの特徴量を得る。Patch Partition では、 $H \times W \times 3$ の RGB 画像を重複しないパッチに分割する。パッチサイズのデフォルトは 4×4 となり、この値は可変である。分割したパッチを Linear Embedding(カーネルサイズ 4×4 、ストライド 4 の畳み込み) を用いて各パッチの特徴量を得る。

3.4.2 Patch Merging

Patch Merging では、各 Stage で得た特徴マップサイズを小さくする。Patch Merging ではパッチ特徴量を各色で表現し、色で表された近傍 2×2 のパッチをチャンネル方向に結合し、サイズを半分にした特徴マップを得る。たとえば $\frac{H}{4} \times \frac{W}{4} \times C$ の特徴マップを Merging すると、サイズを半分にした $\frac{H}{8} \times \frac{W}{8} \times 4C$ の特徴マップを得られる。その後最終的には得た特徴マップのチャンネル次元を $2C$ になるよう全結合層を適用する。

3.4.3 Swin Transformer Block

Swin Transformer Block では、Self-Attention でパッチ間の対応関係を捉えるため、Shifted Window と呼ばれる方法を用いる。Figure 3.5 に Shifted Window の概略図を示す。パッチを広く分割した Window 内のみ Self-Attention で計算することで、計算量を削減できる。

W-MSA は偶数番目の層で適用され、Figure 3.5 左が Self-Attention の範囲を示している。Self-Attention の計算量は、クエリ、キー、バリュー、線形変換を行うための全結

合層4つと、クエリとキー間の行列積である。そのため、 $h \times w$ 個のパッチが存在する Self-Attention 全体の計算量は以下ようになる。

$$\Omega(\text{MSA}) = 4hwC^2 + 2(hw)^2C \quad (3.1)$$

ここで第1項が4つの全結合層、第2項が行列積にかかる計算量である。W-MSA では、 $M \times M$ (Figure 3.5 の例では 2×2) の Window に特徴マップサイズを分割し、Window 内のパッチでのみ Self-Attention で計算する。そのため、Self-Attention 全体の計算量は以下ようになる。

$$\Omega(\text{W-MSA}) = 4hwC^2 + 2M^2hwC \quad (3.2)$$

hw が 196、 C が 16、 M が 7 とすると、 $\Omega(\text{MSA})$ の計算量を式 (3.1) に代入して求めると 1,430,016 となる。一方で $\Omega(\text{W-MSA})$ にかかる計算量は 508,032 となり、W-MSA を取り入れることで計算量は約 1/3 に削減することができる。

SW-MSA は奇数番目の層で適用され、Figure 3.5 右が Self-Attention の範囲を示している。SW-MSA は特徴マップ上の Window を $\frac{M}{2} \times \frac{M}{2}$ だけ移動させた W-MSA である。W-MSA と交互に適用することで、隣接した Window 間でパスがつながる。しかし、Figure 3.5 右のような構造では Window 数が多くなり計算量が増加してしまう。そこで、特徴マップを分割した Window を右下へ 2×2 移動する Cyclic shift を適用することで、パッチ間の対応関係を効率的に計算する。Cyclic shift の概略図を Figure 3.6 に示す。具体的には、複数の Window がある場合、特定の箇所をマスクして Window 間の Attention Weight を 0 にする。Figure 3.6 の例では、左上の Window はマスクせず、右下の Window は各領域 (A、B、C、それ以外の領域) で表されるパッチ間を計算しないようにマスクする。各 Window で Self-Attention を計算したあと、Reverse Cyclic shift で元の特徴マップの位置に戻す。

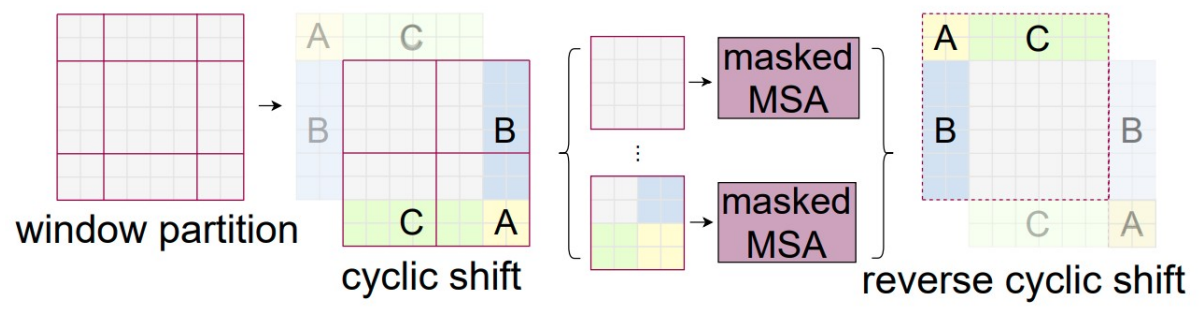


Figure 3.6 Cyclic shift.¹⁾

第4章 実験

4.1 実験目的

Swin Transformer は、Patch Partition を行う際にデフォルトでは 4×4 の正方形にパッチを分割する。このパッチサイズはユーザーが指定して好みのサイズに変更することができる。また、計算量の観点で考えると、パッチサイズを大きくすることで計算量を少なくすることができる。そこで本研究では、デフォルトよりも高い精度を得ることができ、かつ早く計算を終えることができるパッチサイズがあるかどうかを実験を通して検証する。

4.2 実験方法

プログラミング言語として Python を使用し、機械学習ライブラリである Pytorch を利用して学習を実行する。Swin Transformer のモデルは、構築済みモデルをダウンロードして使うことができるライブラリの Pytorch Image Models(timm) によって配布されている、“swin_base_patch4_window7_224.ms_in22k”を使用する。モデルの詳細は 4.5 節で述べる。

入力画像のデータセットは Food101 と呼ばれる食べ物の画像が 101 種類、各 1000 枚入ったデータセットを使用する。今回のモデルは訓練済みモデルではないモデルを用いるため、分類精度が低くなってしまうという観点から 101 種類のうち 10 種類だけを用いて 10 クラス分類を行うこととした。今回学習用に用いた画像のうちの一部を Figure 4.1 に示す。図のように、入力画像は画像内の食べ物が占める割合は大小様々である。また、画像の 75% を訓練用データ、残りの 25% を評価用データとし、すべての訓練用データを 1 度学習し、推論を行った後に評価用データでも推論を行うことを繰り返して学習を進めていった。また、今回使用したモデルは入力サイズが 224×224 であるため、すべての画像をリサイズして入力し、さらに Data Augmentation を施している。

以上の条件下で Swin Transformer が画像を分割するパッチのサイズを 2×2 から 8×8 まで変化させて実行していき、分類精度と計算速度の 2 つの観点からパッチサイズが及ぼす影響について評価する。

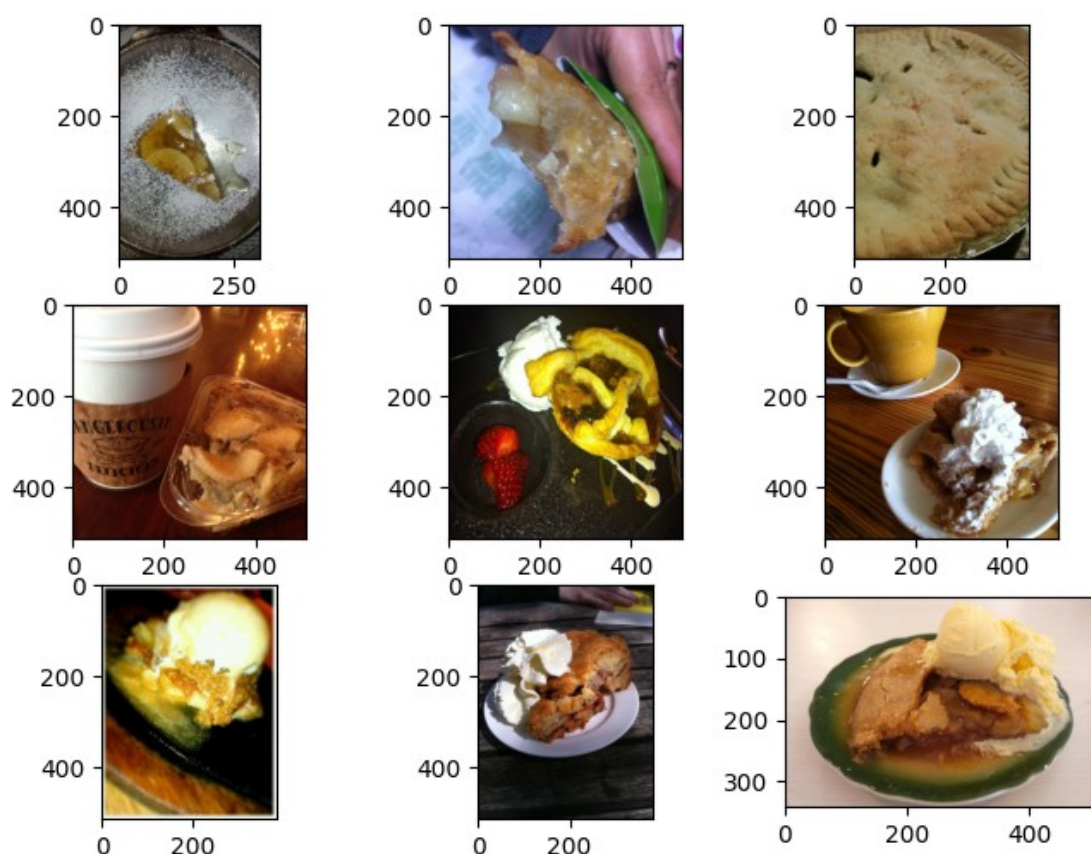


Figure 4.1 Part of input image.

4.3 評価方法

分類精度の観点では、学習が終了した後に出力する Accuracy のグラフから、学習後半の一定の値から上がらないおよその正解率で評価する。計算速度の観点では、1 エポック目が開始する直前から最終エポックの学習終了までの時間を time メソッドを用いて測定し、その値で評価する。

4.4 環境設定

実験を行った環境は、以下の通りである。

- チップ: Apple M1 Max
- メモリ: 64GB
- OS: macOS Sequoia Ver.15.1
- 使用ソフトウェア・ライブラリ: Python 3.9.15, Pytorch 2.4.1

Layer (type:depth-idx)	Output Shape	Param #
SwinTransformer	[16, 10]	--
└PatchEmbed: 1-1	[16, 56, 56, 128]	--
└└Conv2d: 2-1	[16, 128, 56, 56]	6,272
└└LayerNorm: 2-2	[16, 56, 56, 128]	256
└Sequential: 1-2	[16, 7, 7, 1024]	--
└└SwinTransformerStage: 2-3	[16, 56, 56, 128]	--
└└└Identity: 3-1	[16, 56, 56, 128]	--
└└└Sequential: 3-2	[16, 56, 56, 128]	398,344
└└SwinTransformerStage: 2-4	[16, 28, 28, 256]	--
└└└PatchMerging: 3-3	[16, 28, 28, 256]	132,096
└└└Sequential: 3-4	[16, 28, 28, 256]	1,583,120
└└SwinTransformerStage: 2-5	[16, 14, 14, 512]	--
└└└PatchMerging: 3-5	[16, 14, 14, 512]	526,336
└└└Sequential: 3-6	[16, 14, 14, 512]	56,807,712
└└SwinTransformerStage: 2-6	[16, 7, 7, 1024]	--
└└└PatchMerging: 3-7	[16, 7, 7, 1024]	2,101,248
└└└Sequential: 3-8	[16, 7, 7, 1024]	25,203,264
└LayerNorm: 1-3	[16, 7, 7, 1024]	2,048
└ClassifierHead: 1-4	[16, 10]	--
└└SelectAdaptivePool2d: 2-7	[16, 1024]	--
└└└FastAdaptiveAvgPool: 3-9	[16, 1024]	--
└└└Identity: 3-10	[16, 1024]	--
└└Dropout: 2-8	[16, 1024]	--
└└Linear: 2-9	[16, 10]	10,250
└└Identity: 2-10	[16, 10]	--
Total params: 86,770,946		
Trainable params: 86,770,946		
Non-trainable params: 0		
Total mult-adds (G): 2.84		
Input size (MB): 9.63		
Forward/backward pass size (MB): 4970.25		
Params size (MB): 346.76		
Estimated Total Size (MB): 5326.65		

Figure 4.2 Network Details.

4.5 モデル詳細

前述の通り、画像分類を行うモデルは“swin_base_patch4_window7_224.ms_in22k”を使用する。ネットワーク構造は 3.4 節の Figure 3.4 と同様のものである。実行プログラム上で、Torchinfo ライブラリ上の summary メソッドを用いて層ごとの特徴マップの大きさの変遷とパラメータ数を示した情報を表示した。表示結果を Figure 4.2 に示す。

モデルのハイパーパラメータは次のとおりとした。ここに示したパラメータは全て実験を通して変化させていないものである。

- 学習率:0.0001
- エポック数:100
- バッチサイズ:16
- 入力画像:224 × 224, 3 チャンネル

- Num_classes:10
- Window_size:7
- Dropout_rate:0.4

4.6 Data Augmentation

今回の実験では、学習データに Data Augmentation を適用している。オンライン拡張を行っているため、同じ元画像でもエポックごとに微妙に変化を加えた画像を入力している。今回適用した Data Augmentation は以下の通りである。

- 50%の確率で左右反転する。
- -35° から 35° の間の角度にランダムで回転する。
- 50%の確率で、画像の鮮明度を2倍にする。

4.7 実験結果 (分類精度)

この節では、分類精度の観点での実験結果を示す。

4.5 節の設定で、パッチサイズを 2×2 から 8×8 まで変化させて実行していった。まずはデフォルトである 4×4 のパッチサイズで実行した結果を示す。Accuracy のグラフを Figure 4.3、Loss のグラフを Figure 4.4 に示す。この結果から、デフォルトでは Train Accuracy が 0.60 程度、Validation Accuracy が 0.55 程度の精度で分類ができることがわかる。今回の実験は 100 エポックまでの学習であるが、60 エポック目付近から正解率の上昇は止まっていることがわかるため、これ以上回数を重ねても正解率が上がらないと予想される。また、損失の値は Train が約 1.2、Validation が約 1.4 付近で上下している。この値は比較的高い値だと言えるが、正解率自体が 60% 程度であり正解とは異なった判断をしている画像の割合も大きいため、誤差が大きく出てしまっているのは自然である。

続いて、パッチサイズを 2,6,8 と変化させていって同様に実験を進めた。パッチサイズ 2 のときの結果を Accuracy、Loss をそれぞれ Figure 4.5、Figure 4.6 に示す。同様にパッチサイズ 6 の Accuracy を Figure 4.7、Loss を Figure 4.8、パッチサイズ 8 の Accuracy を Figure 4.9、Loss を Figure 4.10 に示す。パッチサイズが 2 のとき、Accuracy のグラフからパッチサイズが他の値のときと比較して学習中の Accuracy の値の上下が激しいことが明らかである。同様に Loss も上下が激しくなっており、特に 80 エポック目付近で大きく Loss が上がっている。更に最終的な Accuracy も 0.50 付近で収束していることがわかり、

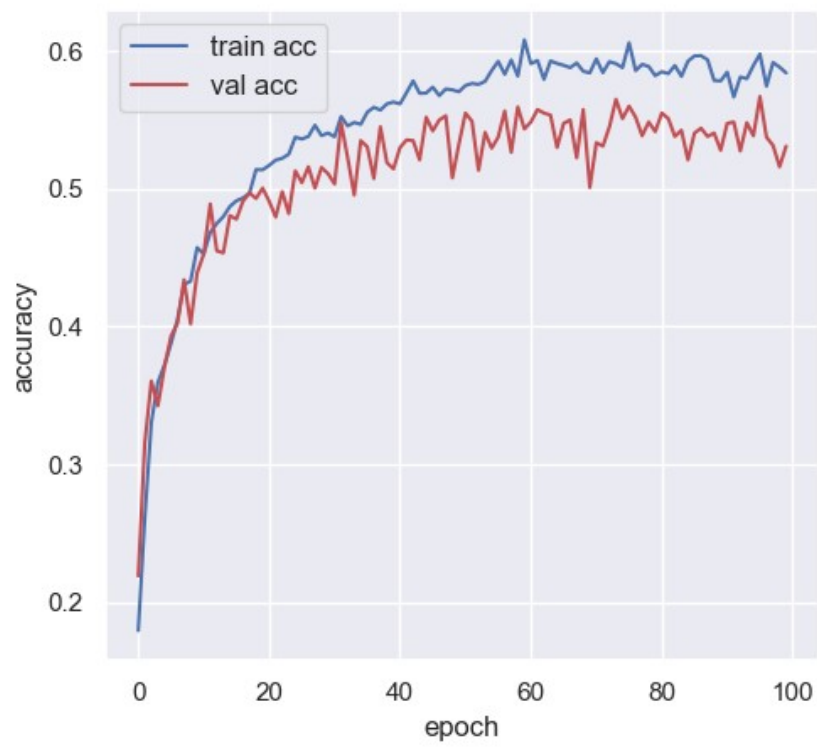


Figure 4.3 Accuracy when patch size is 4.

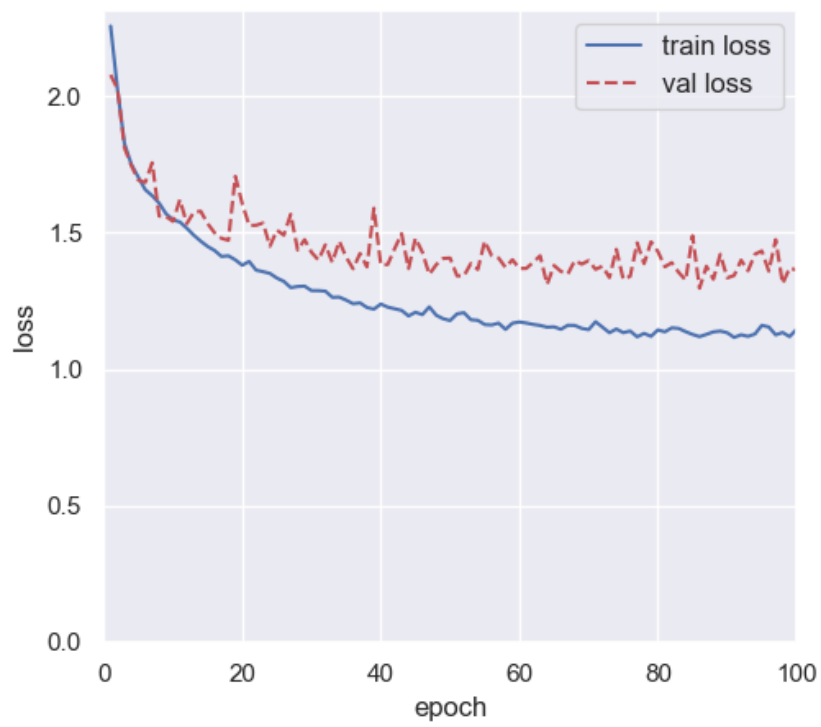


Figure 4.4 Loss when patch size is 4.

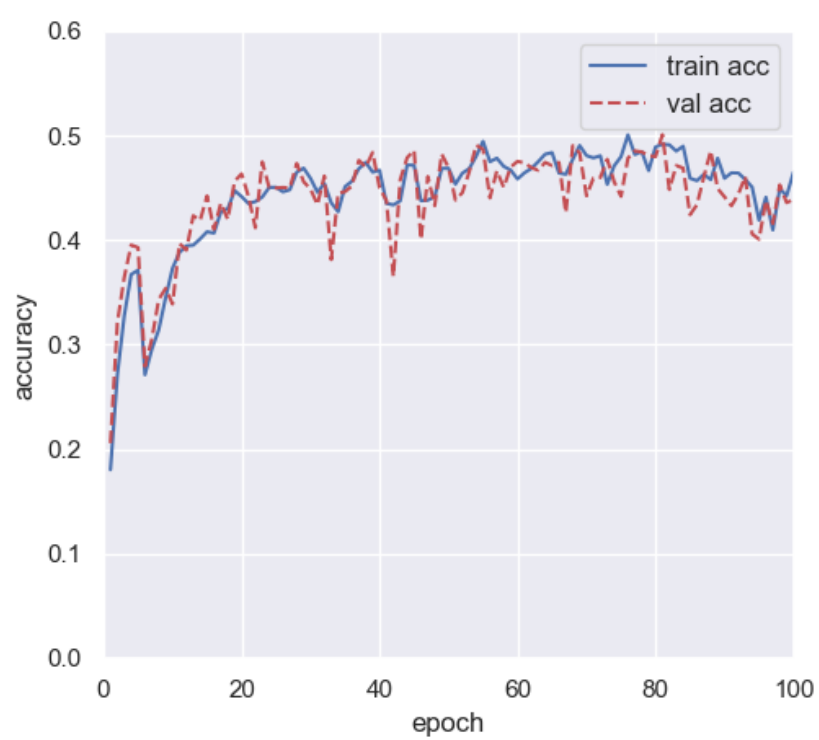


Figure 4.5 Accuracy when patch size is 2.

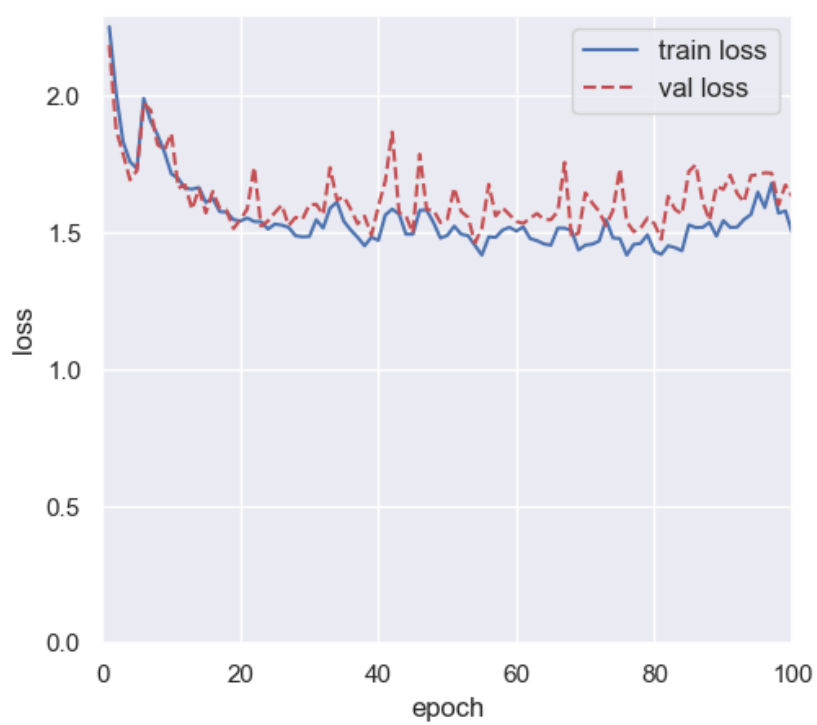


Figure 4.6 Loss when patch size is 2.

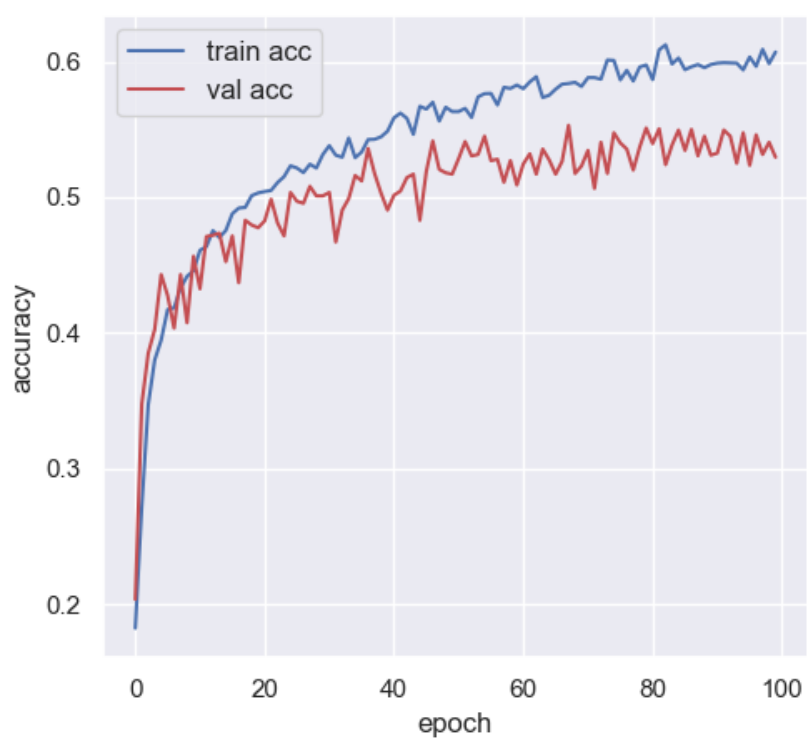


Figure 4.7 Accuracy when patch size is 6.

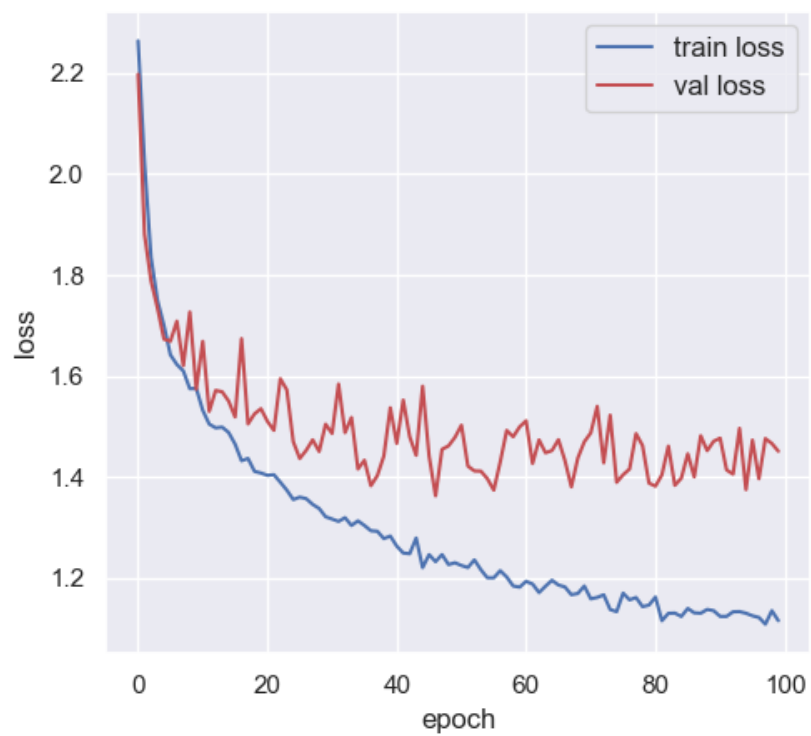


Figure 4.8 Loss when patch size is 6.

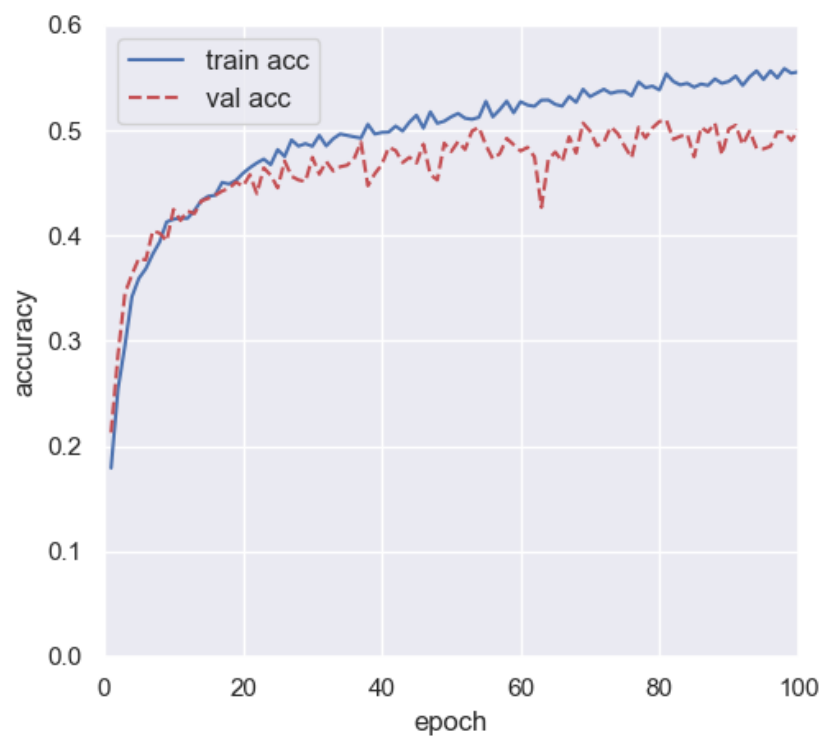


Figure 4.9 Accuracy when patch size is 8.

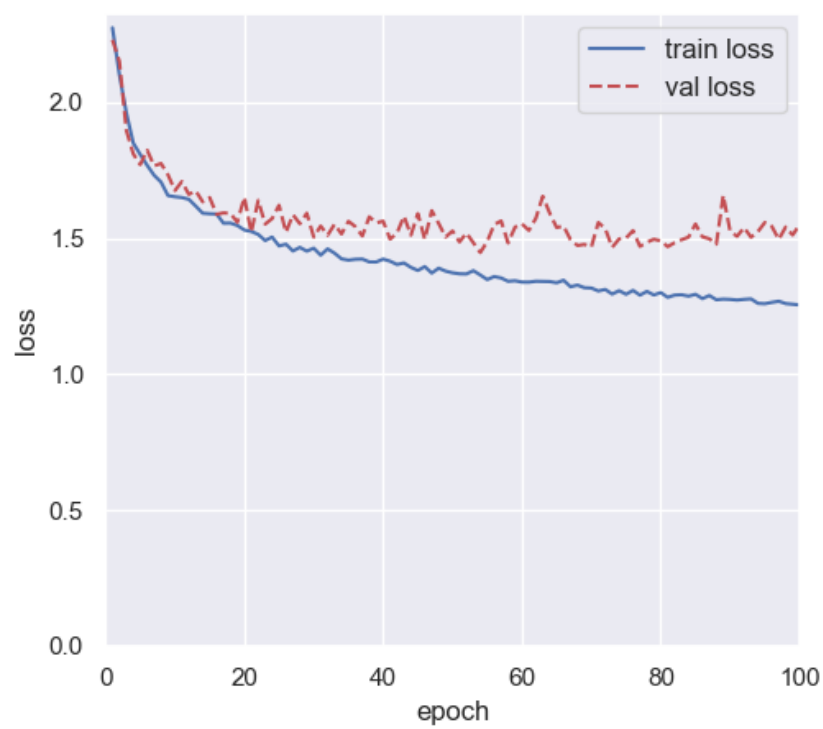


Figure 4.10 Loss when patch size is 8.

Table 4.1 Change in Accuracy as patch size changes.

Patch size	Acc(Train)	Acc(Val)
2	0.51	0.48
3	0.48	0.48
4	0.59	0.52
5	0.49	0.50
6	0.60	0.53
7	0.55	0.49
8	0.55	0.50

他のパッチサイズの時よりも正解率が著しく下がっていることがわかる。また、パッチサイズが6のとき、最終的な Accuracy は Train が 0.60、Validation が 0.55 程度と、パッチサイズが4のときと非常に似た結果であった。

前述した実験も含めた、パッチサイズを2から8まで変化させた際の Accuracy の概算値を Table 4.1 に示す。このように、Accuracy はパッチサイズが4または6のときに限って他の値のときに比べて5%~10%ほど高くなり、逆にパッチサイズが2,3のときは非常に低く、かつ学習中の値の上下が激しくなった。それ以外の値の場合は55%付近になることがわかった。

4.8 実験結果 (計算速度)

パッチサイズを2~8まで変化させたときの、実行時間の推移を Table 4.2 に示す。また、グラフにプロットしたものを Figure 4.11 に示す。グラフより、離散的な値ではあるが実行時間はパッチサイズが大きくなるにつれて負の指数関数的に単調減少していることがわかる。

4.9 考察

パッチサイズは、正方形を1つのベクトルとしてまとめるときの画素の多さである。つまり、パッチサイズが大きいほど広い範囲の情報が1つのベクトルにまとめられ、小さいほど狭い範囲の情報がまとめられるということである。パッチサイズが2や3のときに

Table 4.2 Change in run time as patch size changes.

Patch size	Run Time
2	23h59m03s
3	13h40m36s
4	07h35m59s
5	06h30m24s
6	05h44m44s
7	04h50m48s
8	02h56m42s

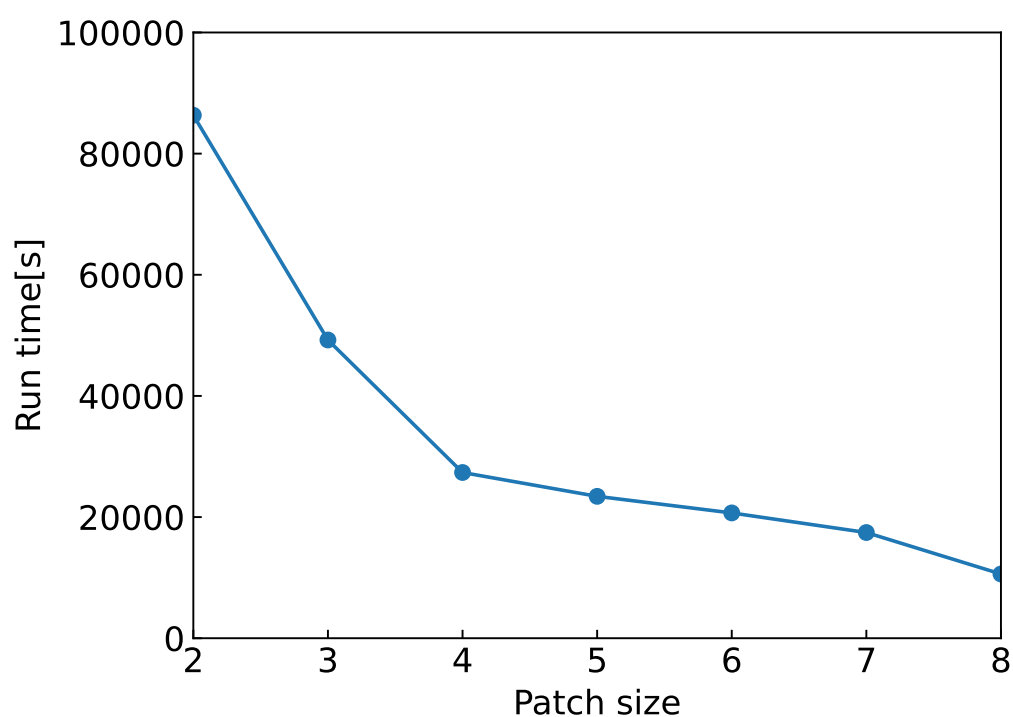


Figure 4.11 Graph of change in run time as patch size changes.

Accuracy が大きく下がった要因は、まとまっている情報が狭すぎて特徴が捉えきれないという点であると考えられる。特に画像内の対象物が大きかった場合にほとんど特徴を得られず入力画像に大きく左右されてしまうことから Accuracy が激しく変化してしまったのだと予想できる。逆にパッチサイズが7や8のときにも Accuracy が下がってしまっ

ているのもひとまとめにする情報が広すぎて特徴を大雑把にしか捉えられないためだろう。Table 4.1 より、パッチサイズが奇数の場合には、偶数の場合に比べると主に Train の Accuracy が大きく下がる特徴がみられる。パッチサイズを奇数にすると、SW-MSA の際に W-MSA のパッチ分割とは異なり不均一になってしまう恐れがあったり、境界部分のトークンがずれる可能性があり、特徴量の抽出にあたって偏りが生じてしまうなど、正しく機能しなくなってしまう可能性がある。以上の考察から、パッチサイズを 4 や 6 にすることで、どんな画像に対してもある程度特徴を捉えられるようなモデルとなるために Accuracy が高くなったのだろうと考えられる。今回の実験は 8 までで行ったが、同様の理由でこれ以上大きくしても Accuracy は下がる一方であることが予想できる。

計算速度の観点では、Swin Transformer がパッチをまとめた Window 単位での Self-Attention 計算を行う特性上、パッチサイズが大きくなって 1 つのベクトルの情報量が上がることで計算に時間がかかるかと予想していたが、予想に反してパッチサイズが大きいほど計算は早く終了することがわかった。これは、1 つの Window の Self-Attention を計算するのにかかる時間のほうが圧倒的に長く、パッチサイズが小さくなることで Window 数が多くなり、計算に時間がかかったのだと考えられる。

また、Swin Transformer は高い精度を誇るモデルである割に 10 クラス分類で 50% 程度と高くない精度であったことについては、主に学習させるデータ量が非常に少なかったことが原因として挙げられるだろう。今回使用したモデルには学習済みモデルが存在し、そちらは ImageNet 呼ばれる大規模データセットを基準に学習を行っており、学習に用いられている画像の枚数は 1400 万枚以上となっている。この学習済みモデルを使用して今回の実験と同じ学習をしてみると、95% の精度を誇った。Transformer 系統のモデルは通常よりも更に多く入力するトークンが必要であり、Swin Transformer も例外ではないため、更に多くの画像を入力できれば精度の点は問題ないと考えている。

第5章 結論

Swin Transformer のデフォルトのパッチサイズは4であるが、今回の実験では、Swin Transformer のパッチサイズを2~8まで変化させ、画像分類の精度と計算速度の2つの観点からパッチサイズが学習に与える影響について研究を行った。結果としてはパッチサイズが6の場合でも、デフォルトの4の場合と同様または更に高い精度を出す可能性があることがわかった。計算速度の実験からわかるように、パッチサイズが4の場合よりも6の場合の方が2時間早く学習を終えられていることから、より精度良く、かつ早く学習することができるのではないかと考えられる。更に学習させる画像の数を増やすことで精度の向上を図る場合にはこの計算速度の差はより大きく広がるため、学習効率をより高める1つの手段として実用段階でも活用できるだろう。

しかし、今回の実験は入力画像を 224×224 に限っていることや、実用上よく使われるであろう学習済みのモデルを用いていないこと、実験を複数回行っていないことなど、様々な不確定要素があったため、この結果を全てのデータセットやモデルに適用することはできない。更に汎用的な性能向上を目指すには、実験手法の再検討や他のパラメータにも目を向けることが必要である。

参考文献

- 1) Ze Liu, Yutong Lin, et al. "Swin Transformer: Hierarchical Vision Transformer Using Shifted Windows", ICCV, p.10012-10022, 2021
- 2) 木島博正, 木島祥子: ニューロンとシナプスにおける情報の変換と伝達, 日本物理学会誌, 47 巻 4 号, p.277-284 (1992)
- 3) 我妻幸長, はじめてのディープラーニング 2, SB クリエイティブ, 2020
- 4) CVML エキスパートガイド, 活性化関数, <https://cvml-expertguide.net/terms/dl/layers/activation-function> (2025/01/19 閲覧)
- 5) CVML エキスパートガイド, CNN の損失関数 その (1): 交差エントロピーと MSE, <https://cvml-expertguide.net/terms/dl/loss-function/loss-function-part1/> (2025/01/31 閲覧)
- 6) AISmiley, 過学習とは? 具体例と発生する原因・防ぐための対策方法をご紹介します, https://aismiley.co.jp/ai_news/overtraining/ (2025/02/06 閲覧)
- 7) aws, オーバーフィットとは何ですか?, <https://aws.amazon.com/jp/what-is/overfitting/> (2025/02/06 閲覧)
- 8) 片岡裕雄, 山本晋太郎, 徳永匡臣, 箕浦大晃, 品川政太郎, QIU YUE, Vision Transformer 入門, 技術評論社, 2022

謝辞

本研究を進めるに当たり、ご多忙の中にもかかわらず多大なご指導を賜りました出口利憲先生に深く感謝するとともに、同研究室内でともに勉学に励んだ瀬瀬翔氏、古田陵矢氏、水野良亮氏に厚く御礼を申し上げます。