卒 業 研 究 報 告 題 目

転移学習とファインチューニングによる画像分類

Image Classification by Transfer Learning and Fine-Tuning

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

2020E17 纐纈　翔

令 和 7 年（ 2 0 2 5 年 ）　　2 月 1 7 日 提 出

Abstract

In recent years, artificial intelligence has developed significantly and is being used to create a variety of systems. Systems that automatically identify images and sort products have also been introduced.

In this study, we will explore efficient AI learning methods using transfer learning and fine tuning. In detail, we will collect data sets of similar types, such as dogs and cats, roses and tulips, and perform transfer learning and fine tuning using a pre-trained MobileNet V2 model. The ratio of the number of times of transfer learning and fine tuning is varied to determine the most efficient ratio.

The results showed that starting fine-tuning early and increasing the number of fine-tuning sessions increased the efficiency of learning. Therefore, it was concluded that fine tuning should be the main axis of learning when using MobileV2 net as a pre-trained model for transfer learning.

However, changing the pre-trained model will significantly change the results, so if this study is continued in the future, similar experiments should be conducted with different models.

目 次

第1章　序論

近年、人工知能 (AI) が目まぐるしく発展を遂げており、AI の利便性が注目されつつある。現在でも、工場の生産ラインで流れてくる製品の種類を自動で分類・振り分けをするシステムや、空港でパスポートと顔写真を AI が学習して顔認証をするシステムなどが導入されている。

先行研究においては、画像分類システムの構築や分類の精度向上について多くの結果が出ている。例としては、CNN を用いた特定魚種の識別精度向上のための研究 [1] などが挙げられる。しかし、AI の学習において学習時間の短縮や必要なデータ量を減らすというのも、考慮すべき課題であると考えられる。このような課題を解決するため、転移学習やファインチューニングといった手法が活用される。これらの手法を活用することで、先に述べた課題を解決するだけでなく応用範囲の広い AI 学習を行うことができる。具体例としては、動物の識別モデルを自動車や船などの異なる分野の識別にも活用できるようになるということである。

本研究では、転移学習とファインチューニングを用いて事前学習済み畳み込みニューラルネットワークから基本モデルを作成し、多くの分野に対して素早く学習を終えることのできるモデルの作成を試みる。具体的な手法としては、転移学習後にファインチューニングを行い、これらの回数の比率を変化させていき、効率の良い学習ができる比率を調査する。これにより、さまざまなシステムの構築をスムーズに行うことに貢献することができると考える。

第2章　ニューラルネットワーク[2)]

2.1　神経細胞ネットワーク

神経細胞は、生物の神経系において情報の伝達や記憶の保持を担っていると考えられている。人の脳には、このような神経細胞が約 1000 億程度存在すると考えられている。この神経細胞における情報伝達を Figure 2.1 に示す。

神経細胞において、樹状突起 (dendrite) はその名の通り木のように枝分かれしており、他の神経細胞からの情報を受け取る。樹上突起は複数の神経細胞の軸索端末 (axon terminal) と接続されており、他の神経細胞からの信号が樹状突起に伝わると、細胞の電位が上昇する。この電位が複数の入力によりある値を超えると、神経細胞が興奮して信号が発生し、軸索 (axon) を通って枝分かれした軸索端末まで伝わる。軸索端末は、他の神経細胞の樹状突起、もしくは筋肉などの出力器官と、シナプス (synapse) と呼ばれる接続部で接合されている。

シナプスの前の神経細胞 (シナプス前細胞) と、シナプスの後の神経細胞 (シナプス後細胞) の接合部には 20 nm 程度の間隙があり、神経伝達物質と呼ばれる化学物質がこの間隙を通過することで神経細胞間の情報伝達が行われる。軸索端末に信号が来ると、神経伝達物質を内包するシナプス小胞から、シナプス間隙に神経伝達物質が放たれる。神経伝達物質は、シナプス後細胞と受容体と結合し、シナプス後細胞内の電位が変化する。

シナプスの伝達効率は、前後の神経細胞の活動状態などによって変化し、保持される。このようなシナプスの伝動効率の変化は、記憶や学習において重要な役割を果たしていると考えられている。

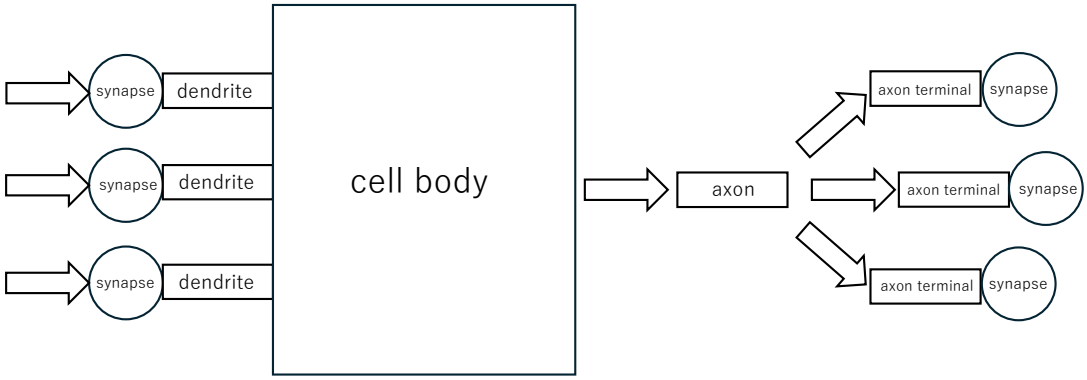


Figure 2.1　Schematic diagram of information transmission in a neuron.

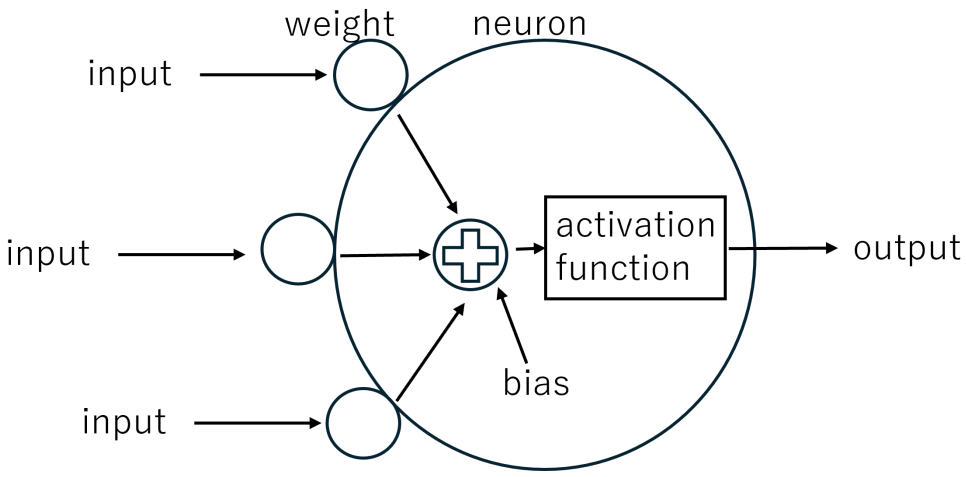


Figure 2.2　Model of neuron.

2.2　ニューロンのモデル化

コンピューター上のモデル化された神経細胞のことを人工ニューロン (Artificial Neuron)、コンピューター上のモデル化された神経細胞ネットワークのことを人工ニューラルネットワーク (ANN、Artificial Neural Network) と呼ぶが、これ以降は簡単にするためにニューロン、ニューラルネットワークという名称を使う。

個々の神経細胞は、簡単な演算能力しか持たないと考えられている。それでも、このシンプルなユニットがお互いにつながり連動することで、高度な認識・判断能力が生まれる。

ニューラルネットワークの場合も同様に、ここのニューロンで行われている演算はシンプルなものである。しかしながら、多数のニューロンがつながり合うことで高度な認識・判断能力が発揮される。個別のニューロンのモデルを Figure 2.2 に示す。

ニューロンには複数の入力があるが、出力は 1 つだけである。これは、樹状突起への入力が複数あるのに対して、軸索からの出力が 1 つだけであることに対応する。各入力には、重みを掛け合わせる。重みは結合荷重とも呼ばれ、入力ごとに値が異なる。この重みの値がシナプスにおける伝達効率が相当し、値が大きければそれだけ多くの情報が流れることになる。

そして、入力と重みを掛け合わせた値の総和に、バイアスと呼ばれる定数を足す。バイアスは言わば、ニューロンの感度を表す。バイアスの大小により、ニューロンの興奮しやすさが調整される。

入力と重みの積の総和にバイアスを足した値は、活性化関数と呼ばれる関数で処理さ

れ、ニューロンの興奮状態を表す信号に変換される。この信号がニューロンの出力である。活性化関数は、言わばニューロンを興奮させるための関数である。

ここから、このニューロンのモデルを数式で表現する。まず、入力と重みの積を表現する。w が重み、x がニューロンへの入力とすると、入力と重みの積は次のように表すことができる。この場合、w、x ともにスカラーである

$$xw \tag{2.1}$$

次に、入力と重みを掛け合わせたものを、1 つのニューロンのすべての入力で足し合わせる。各入力、各重みはそれぞれ異なる値をとるため、その積の総和は入力の数を n とすると次の式で表すことができる。添字 k はそれぞれの入力を表す。

$$\sum_{k=1}^{n} x_k w_k \tag{2.2}$$

次に、入力と重みの積の総和にバイアス b を加える。これを u で表すと、次の式を得る。

$$u = \sum_{k=1}^{n} (x_k w_k) + b \tag{2.3}$$

この u を、活性関数に入力する。

活性化関数を f、ニューロンからの出力を y であらわすと、活性化関数と出力の関係は次の式であらわされる。

$$y = f(u) = f\left(\sum_{k=1}^{n} (x_k w_k) + b\right) \tag{2.4}$$

2.3 ニューロンのネットワーク化

ニューロンを複数接続しネットワーク化することで、ニューラルネットワークが構築される。このニューラルネットワークのモデル図を、Figure 2.3 に示す。ニューラルネットワークにおける層は、入力層 (input layer)、中間層 (intermediate layer)、出力層 (output layer) の 3 つに分類することができる。入力層はニューラルネットワーク全体の入力を

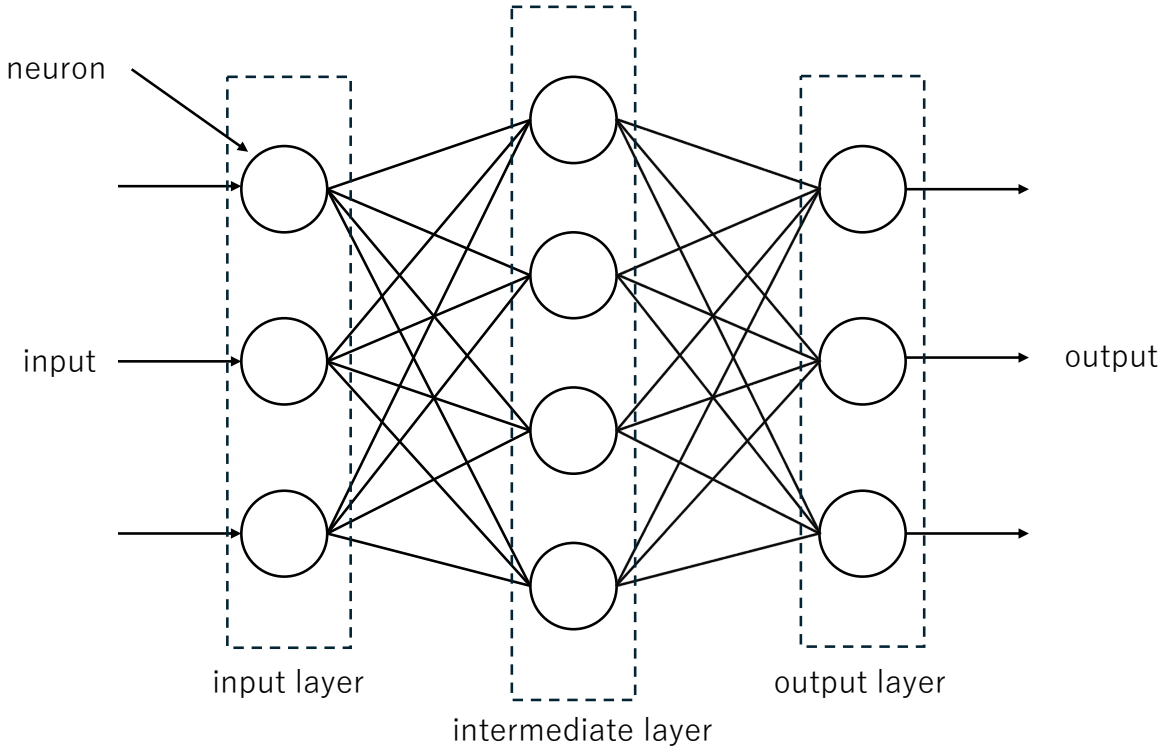


Figure 2.3　Model of neural network.

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、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においては、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。ニューラルネットワークにおいて、入力から出力に向けて情報が伝わっていくことを順伝播という。逆に、出力から入力に向けて情報が遡っていくことを逆伝播という。

ここから、上の層とはネットワークの入力に近い層のことを指す。また、下の層とは出力に近い層のことを指す。上の層のすべてのニューロンは、それぞれ下の層のすべてのニューロンと接続されている。これは、下の層のすべてのニューロンが、それぞれ上の層のすべてのニューロンと接続されている、と表現することもできる。上記の通り、ニューロンへのそれぞれの入力には重みをかける。重みの数は入力の数と等しいので、上の層のニューロン数を m とすると、下の層のニューロンは1つあたり m 個の重みを持つことになる。下の層のニューロン数を n とすると、下の層には合計 $m \times n$ 個の重みが存在することになる。

このような重みだが、例えば上の層の1番目のニューロンから、下の層の2番目のニュー

ロンへの入力の重みは w_{12} と表す。重みは、上の層のすべてのニューロンと、下の層のすべてのニューロンのそれぞれの組み合わせごとに設定する必要があるが、ここでは行列を使用する。以下のような $m \times n$ の行列に、下の層のすべての重みを格納することができる。W は重みを表す行列である。

$$W = \begin{pmatrix} w_{11} & w_{12} & \cdots & w_{1n} \\ w_{21} & w_{22} & \cdots & w_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m1} & w_{m2} & \cdots & w_{mn} \end{pmatrix} \tag{2.5}$$

また、上の層の出力はベクトルで表すことができる。上の層には m 個のニューロンがあるため、ベクトルの要素数は m になる。i を上の層の添字、j を下の層の添字とし、$\vec{y_i}$ を上の層の出力を表すベクトル、$\vec{x_j}$ を下の層への入力を表すベクトルとすると、次のような表記が可能となる。

$$\vec{y_i} = \vec{x_j} = (x_1, x_2, \cdots, x_m) \tag{2.6}$$

上の層の出力は下の層の入力に等しくなる。

バイアスについては前節で解説したが、これもベクトルで表記することが可能である。バイアスの数は下の層のニューロンの数に等しく、下の層のニューロンは n 個であるため、バイアス $\vec{b_j}$ は次のように表すことができる。

$$\vec{b_j} = (b_1, b_2, \cdots, b_n) \tag{2.7}$$

また、下の層の出力の数はニューロンの数 n に等しいので、ベクトル $\vec{y_i}$ を用いて次のように表記することができる。

$$\vec{y_j} = (y_1, y_2, \cdots, y_n) \tag{2.8}$$

入力と重みの積の総和を求める必要があるが、助かることに行列積を用いて一度に求めることができる。式 (2.5) の W は下の層の重みを表す行列で、式 (2.6) の $\vec{x_j}$ は下の層への入力であるため、$\vec{x_j}$ を $1 \times m$ の行列と捉えると、以下の行列積で各ニューロンにお

ける入力と重みの積の総和を求めることができる。

$$\begin{aligned}\vec{x_j}W &= (x_1, x_2, \cdots, x_m)\begin{pmatrix} w_{11} & w_{12} & \cdots & w_{1n} \\ w_{21} & w_{22} & \cdots & w_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m1} & w_{m2} & \cdots & w_{mn} \end{pmatrix} \\ &= \left(\sum_{k=1}^{m} x_k w_{k1}, \sum_{k=1}^{m} x_k w_{k2}, \cdots, \sum_{k=1}^{m} x_k w_{kn}\right)\end{aligned} \tag{2.9}$$

行列積の結果は要素数 n のベクトルである。このベクトルの各要素は、下の層の各ニューロンにおける重みと入力の積の総和になっている。これにバイアス $\vec{b_j}$ を加えたものを $\vec{u_j}$ とするが、$\vec{u_j}$ は次のように求められる。

$$\begin{aligned}\vec{u_j} &= \vec{x_j}W + \vec{b_j} \\ &= (x_1, x_2, \cdots, x_m)\begin{pmatrix} w_{11} & w_{12} & \cdots & w_{1n} \\ w_{21} & w_{22} & \cdots & w_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m1} & w_{m2} & \cdots & w_{mn} \end{pmatrix} + (b_1, b_2, \cdots, b_n) \\ &= \left(\sum_{k=1}^{m} x_k w_{k1} + b_1, \sum_{k=1}^{m} x_k w_{k2} + b_2, \cdots, \sum_{k=1}^{m} x_k w_{kn} + b_n\right)\end{aligned} \tag{2.10}$$

$\vec{u_j}$ の各要素は、重みと入力の積の総和にバイアスを足したものになっている。次に、活性化関数を使用する。ベクトル $\vec{u_j}$ の各要素を活性化関数に入れて処理し、下の層の出力を表すベクトル $\vec{y_j}$ を得る

$$
\begin{aligned}
\vec{y_j} &= (y_1, y_2, \cdots, y_n) \\
&= f(\vec{u_j}) \\
&= f(\vec{x_j}W + \vec{b_j}) \\
&= \left(f\left(\sum_{k=1}^{m} x_k w_{k1} + b_1\right), f\left(\sum_{k=1}^{m} x_k w_{k2} + b_2\right), \cdots, f\left(\sum_{k=1}^{m} x_k w_{kn} + b_n\right)\right)
\end{aligned}
\tag{2.11}
$$

$\vec{y_j}$ の要素数は、下の層のニューロンの数と同じ n になる。$\vec{y_j}$ は、単一ニューロンの際の式 (2.4) を層全体に拡張したものとなっている。さらに下の層がある場合、$\vec{y_j}$ はその層への入力となる。

ニューロンを層として扱うことで、2 つの層間の情報の伝播を数式にまとめることができた。層の数が 3 つ以上になっても、式 (2.11) を用いて層から層へ次々に情報を順伝播することができる。

ニューラルネットワークは、層の数が増えて規模が大きくなれば、より生物に近い柔軟な認識・判断能力を持つことが可能になる。そのためには、各ニューロンの重みとバイアスを自動で調整する仕組みが必要となる。

ニューロン間の重みは、負の値になることがある。これは、生物学的にはどのように解釈すれば良いのか。シナプスには化学シナプスと電気シナプスに分類できる。化学シナプスは神経伝達物質で、電気シナプスはイオン電流により情報の伝達を行う。一般にシナプスという際は、化学シナプスを指すことが多い。

そして、化学シナプスには興奮性シナプスと抑制性シナプスがある。興奮性シナプスは神経細胞を興奮させるが、抑制性シナプスはニューロンの興奮を抑える。したがって、ニューラルネットワークにおける正の値の重みは、興奮性シナプスに対応すると考えることができる。逆に、負の値の重みは抑制性シナプスに対応すると考えると、負の重みを生物学的に解釈することも可能となる。

2.4 分類問題

ニューラルネットワークで扱う問題は、大きく回帰問題と分類問題に分けることができる。本文では分類問題に関する実験を取り扱っているため、分類問題について記述し

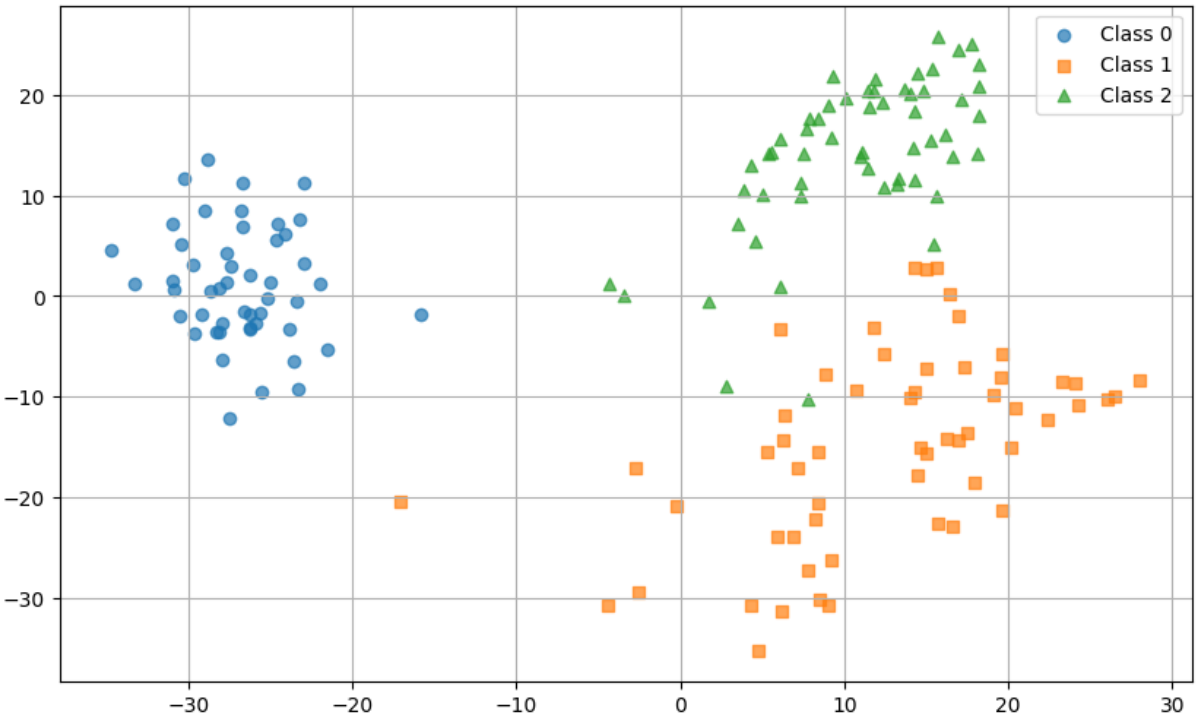


Figure 2.4　Image of classification problem.

ていく。

分類問題は、データを決められた複数の枠に分類する問題のことである。例えば、以下のような問題は分類問題となる。

- 葉の画像から植物を分類する
- 画像に映る人物を男女に分類する
- 体のサイズや特徴からイルカとクジラを分類する
- 手書きのアルファベットを、a から z に分類する

これらの分類結果は、例えば「カエデ」、「女」、「クジラ」、「s」のように離散的な値となる。なお、画像をニューラルネットワークに入力する場合は、画像の各ピクセルが入力となる。詳しくは畳み込みニューラルネットワークの章で解説する。分類問題のイメージを Figure 2.4 で示す。このグラフは、横軸と縦軸の値によりデータを分類しているが、各マーカーが 1 つのグループに相当する。

ニューラルネットワークでこのような分類ができれば、グラフ上に新たに追加する点がどのグループに属するか、ある程度予測することができる。

分類問題を扱う場合、出力層のニューロンは次のようになる。このケースでは、葉の画像から植物をカエデとイチョウとツツジに分類する。分類問題における出力層のニューロンの動きを Figure 2.5 に示す

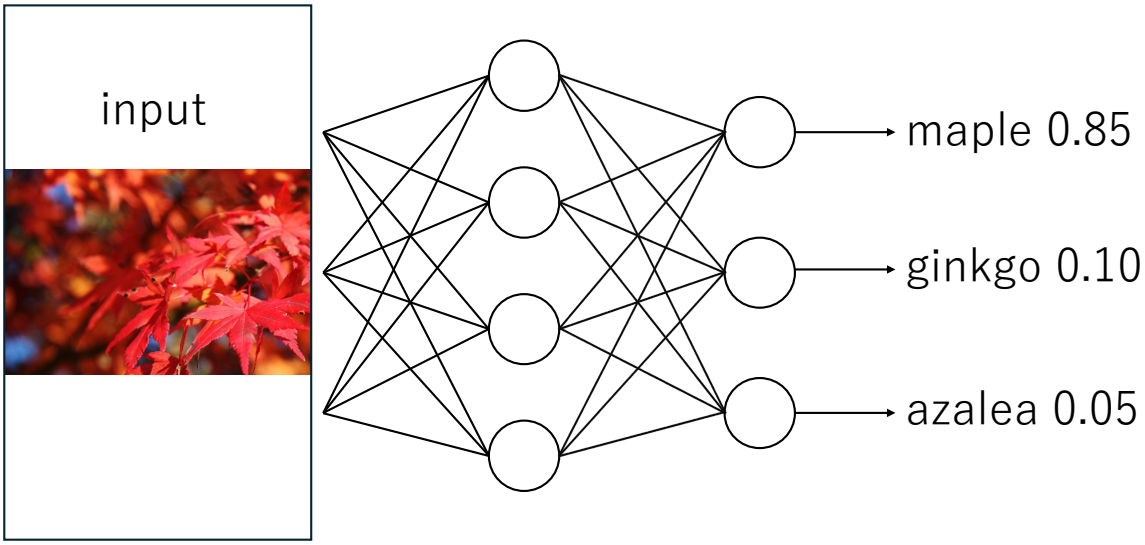


Figure 2.5　Output layer neurons in classification problems.

この場合、分類の枠が 3 つであるため、出力層のニューロンは 3 個になる。これらのニューロンからの出力は、各枠に分類される確率である。この例ではカエデのニューロンからの出力が最も大きく、葉はカエデのものであると判断するのが最も妥当である。

2.5　活性化関数 (ReLU)

以前に少し触れたが、活性化関数は言わばニューロンを興奮させるための関数である。ニューロンへの入力と重みをかけたものの総和にバイアスを足し合わせた統合された値を、ニューロンの興奮状態を表す信号に変換する。活性化関数がないとニューロンの演算は単なる積の総和になってしまい、ニューラルネットワークから複雑な表現をする力が失われてしまう。活性化関数には様々な種類があるが、本節では実験に使用した ReLU 関数を記述する。

ReLU はランプ関数とも呼ばれ、Figure 2.6 のようなグラフで表される。

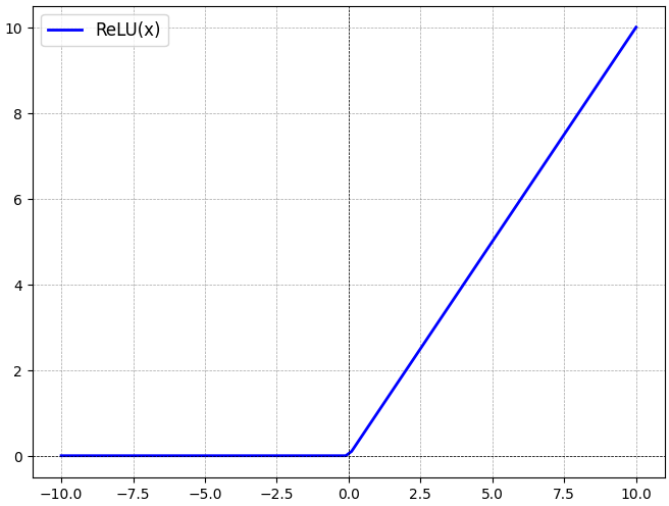


Figure 2.6　Graph of ReLU.

$x > 0$ の範囲でのみ立ち上がるのが特徴的な活性化関数である。ReLU は次のような式で表される。

$$y = \begin{cases} 0 & (x \leq 0) \\ x & (x > 0) \end{cases} \tag{2.12}$$

関数への入力 x が負の場合、関数の出力 y は 0 に、x が正の場合、y は x と等しくなる。ReLU はシンプルであり、なおかつ層の数が多くなっても安定した学習ができるため、最近のディープラーニングでは主にこの ReLU が出力層以外の活性化関数として用いられている。

なお、ReLU の導関数は次の通りになる。

$$y' = \begin{cases} 0 & (x \leq 0) \\ 1 & (x > 0) \end{cases} \tag{2.13}$$

関数への入力 x が負の場合、関数の出力 y' は 0 に、x が正の場合、y' は 1 になる。微分値が x の値によらず安定した値を取るのは、ReLU の大きなメリットである。

第3章　畳み込みニューラルネットワーク (CNN)[2)]

3.1　畳み込みニューラルネットワーク (CNN) の概要

畳み込みニューラルネットワーク (以降は CNN) は、ヒトの視覚のように画像認識を得意としている。Figure 3.1 は CNN の例であるが、CNN は画像を入力した分類問題をよく扱う。この図においては、出力層の各ニューロンの各動物に対応し、出力の値がその動物である確率を表している。

CNN には畳み込み層、プーリング層、全結合層という名前の層が登場する。例えば、猫の写真を学習済みの CNN に入力すると、90 %でネコ、5 %でイヌ、3 %でウサギ、1 %でネズミ、のようにその物体が何である確率が最も高いかを教えてくれる。

CNN には、画像を柔軟に精度良く認識するために、通常のニューラルネットワークとは異なる仕組みが備わっている。畳み込み層では、出力が入力の一部の影響しか受けない局所性の強い処理が行われる。また、プーリング層においては、認識する対象の位置に柔軟に対応できる仕組みが備わっている。

3.2　視覚のメカニズム

CNN は生物の視覚がモデルになっているため、その仕組みについて記述する。人の視覚情報の主な経路を Figure 3.2 に示す。

ヒトの左右の目には網膜があり、網膜は視神経に繋がっている。左右それぞれの目における左側の網膜は右の視野を捉え、右側の網膜は左の視野を捉える。そして、網膜と繋がった視神経は半分が交差し、左側の視野に関する情報は右脳が、右側の視野に関す

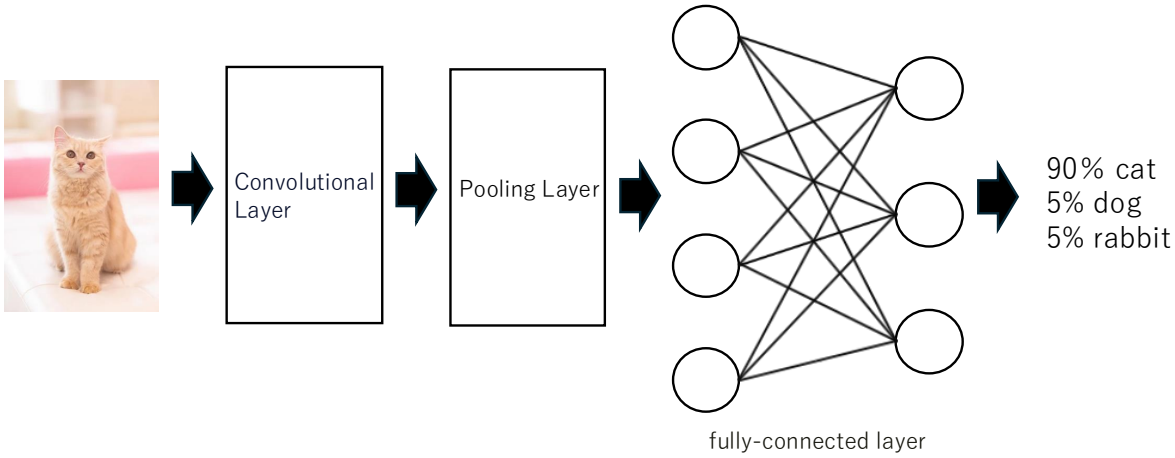


Figure 3.1　Image of CNN.

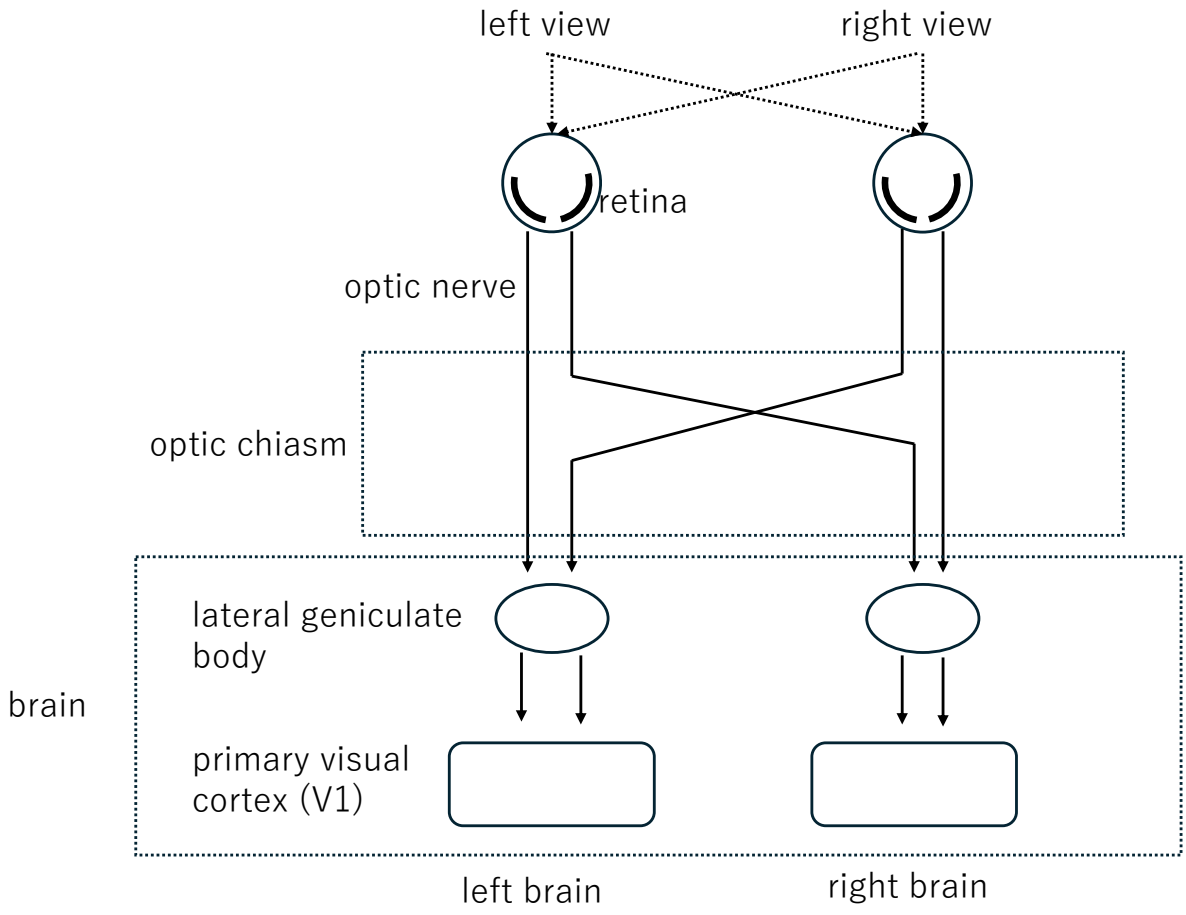


Figure 3.2　Visual information pathway.

る情報は左脳が処理することとなる。

左右それぞれの視野の情報は、脳の視床の一部である外側膝状体で処理された後に、大脳皮質の一番後ろにある一次視覚野 (V1) に届く。一次視覚野で処理された情報は、背側皮質視覚路と腹側皮質視覚路という 2 つの経路に分かれてそれぞれ処理されていく。

目の網膜に存在する、光刺激を電気信号に変換する細胞を視細胞という。視細胞には明所で機能する錐体と暗所で機能する杆体の 2 種類がある。錐体は片目に 500 万個ほど存在するが、視神経における神経繊維は 150 万本ほどしかない。500 万個の視細胞から 150 万本の繊維に情報が伝わるため、網膜ですでに情報が圧縮されていることとなる。

外側膝状体では眼から入った情報の選別が行われていると考えられている。

一次視覚野はよく研究されている脳の領域である。一次視覚野には 1 億 4000 万個ほどの神経細胞が存在すると考えられており、視覚に関する複雑な処理が行われている。それらの神経細胞の中には、網膜の特定の場所に特定のパターンが入力されると興奮し、それ以外の時は興奮しないという選択的な振る舞いを示すものがある。そのような細胞

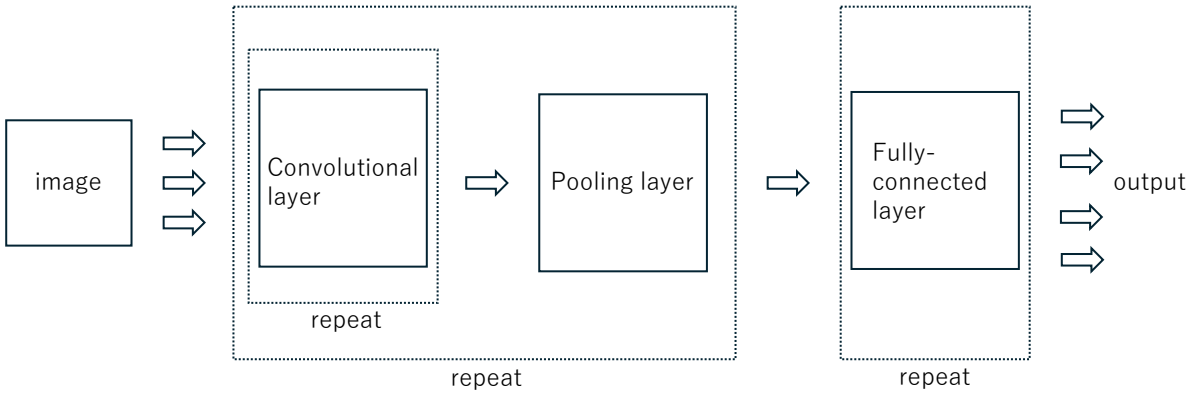


Figure 3.3　Struture of CNN.

には単純型細胞、および複雑型細胞と呼ばれる 2 種類があり、それぞれ性質が異なる。

視覚神経系における個々の神経細胞には、受容野がある。受容野とは、個々の神経細胞が対応する視野の範囲のことで、神経細胞にとっていわば外部に開かれた小さな窓である。ここの単純型細胞はわずかな網膜の範囲に対応する狭い受容野しか持っていないが、視覚野には非常に多くの細胞があるため、各細胞が様々な範囲の視野を受け持つことで視野全体に対応することができる。

単純型細胞は特定の傾きを持った明暗の境界を検出する細胞で、対象の位置の変化に敏感である。また、複雑型細胞は検出されるべき特徴が受容野の中に入ってさえいれば、位置に関わらず興奮する位置の変化に鈍感な細胞である。単純型細胞は CNN における畳み込み層のニューロンに対応し、複雑型細胞はプーリング層のニューロンに対応するが、畳み込みそうとプーリング層については後ほど記述する。

3.3　CNN の構造

ここからは、CNN の構造について、まず概要を記述する。複数の層で構成されている点はこれまで扱ったニューラルネットワークと同様である。CNN の構造を Figure 3.3 に示す。

CNN の場合は、層に畳み込み層、プーリング層、全結合層の 3 種類がある。画像は畳み込み層に入力されるが、畳み込み層とプーリング層は何度か繰り返されて、全結合層につながる。全結合層も何度か繰り返されて、最後の全結合層は出力層になる。

畳み込み層では、入力された画像に複数のフィルタで処理を行う。フィルタ処理の結果、入力画像は画像の特徴を表す複数の画像に変換される。そして、プーリング層では画像の特徴を損なわないように画像のサイズが縮小される。全結合層では、これまで扱っ

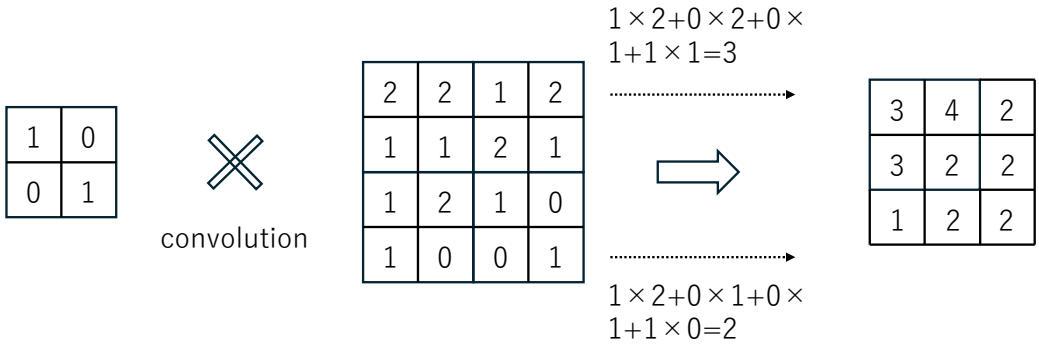


Figure 3.4　Process of convolution.

てきたニューラルネットワークの層と同じように、層間のすべてのニューロンが接続される。

全結合層は、通常のニューラルネットワークで使われる層と同じものである。

3.4　畳み込み層

畳み込み (convolution) は、画像処理の分野ではポピュラーな演算である。画像に対して畳み込みを行うことで、画像にある特徴を強めたり弱めたりすることができる。畳み込み層では、この畳み込み処理により入力画像をより特徴が強調されたものに変換する。

また、画像には局所性という性質がある。局所性とは、各ピクセルが近傍のピクセルと強い関連性を持っている性質のことである。例えば、ある隣り合ったピクセル同士は似たような色になる可能性が高く、複数のピクセルからなるかたまりが物体の輪郭を構成することもある。畳み込み層は、このような画像の局所性を利用して画像の特徴を検出する。

畳み込み層では、複数のフィルタを用いて特徴の検出が行われる。それぞれのフィルタは、それぞれ異なる特徴を検出する。Figure 3.4 に畳み込み層におけるフィルタを用いた畳み込み処理の例を示す。ここでは、各ピクセルの値を数値で表している。この値がピクセルの色合いに相当する。

この図では、わかりやすくするために入力を 4×4 ピクセルの画像とし、フィルタの数は 1 つとしている。フィルタのサイズは 2×2 である。畳み込みでは、フィルタの入力画像の各位置にずらして、重なったピクセル同士の値を掛け合わせる。そして掛け合わせた値をフィルタの位置ごとに足し合わせて新たなピクセルとする。結果として、畳み込みにより 3×3 の新たな画像が生成される。

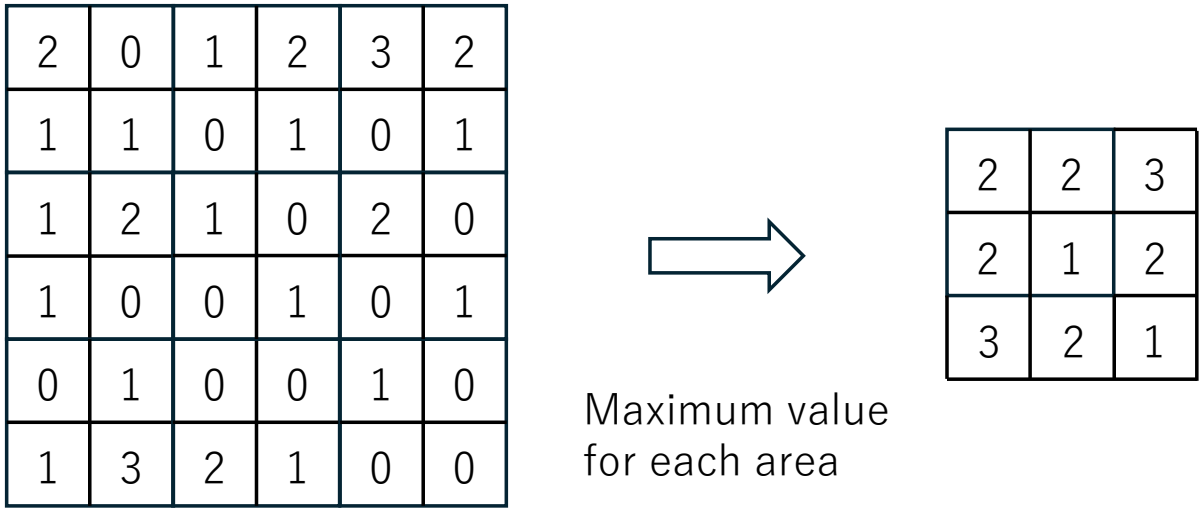


Figure 3.5　Example of pooling.

畳み込みを行うともとの画像よりもサイズが小さくなる。上記の例では 4×4 の画像に 2×2 のフィルタをかけて 3×3 の画像が生成されたが、例えば 3×3 の画像に 2×2 のフィルタをかけると 2×2 に、5×5 の画像に 2×2 のフィルタをかけると 4×4 になる。

フィルタの内容を変えることで、様々な特徴が検出された画像に変換することができる。

3.5　プーリング層

次に、プーリング層について記述する。プーリング層は通常畳み込み層の直後に配置される。プーリング層では画像を各領域に区切り、各領域を代表する値を抽出し並べて新たな画像とする。このような処理をプーリング (pooling) という。プーリングの例を Figure 3.5 に示す。

この場合は各領域の最大値を、各領域を代表する値としている。このようなプーリングの方法を、MAX プーリングという、他にも領域の平均値を取る平均プーリングなどの方法もあるが、CNN では MAX プーリングを使用することが多いため、以降プーリングという言葉は MAX プーリングを指すこととする。

Figure 3.5 で示されているように、プーリングを行うと画像が縮小される。例えば 8×8 ピクセルの画像に対して 2×2 の領域でプーリングすると、画像のサイズは 4×4 になる。

プーリングはいわば画像をぼやかす処理であるため、対象の位置の感度が低下する。これにより、対象の位置が多少変化しても、結果は同じようになる。プーリング層は、一次視覚野における複雑型細胞のように位置の変化に対するロバスト (頑強) 性を与えることになる。また、プーリングにより画像サイズが小さくなるため、計算量が削減される

効果もある。プーリング層で区切る領域は固定であり、学習するパラメータがないため学習は行われないまた、チャンネルが合流したり分岐したりすることはないため、入力のチャンネル数と出力のチャンネル数は同じになる。

3.6 全結合層

全結合層 (fully connected layer) は、通常のニューラルネットワークで用いられる層のことである。全結合層は、通常畳み込み層とプーリング層を何度か繰り返した後に配置される。畳み込み層とプーリング層により抽出された特徴量に基づき、演算を行い、結果を出力する。

全結合層同士の接続では、通常のニューラルネットワークと同様にニューロンは隣り合う層のすべてのニューロンと接続される。畳み込み層やプーリング層の出力を全結合層に入力する場合は、画像を平坦なベクトルに変換する。例えば、出力の画像の高さが H、幅が W、チャンネル数が F である場合、全結合層の入力はサイズが $H \times W \times F$ のベクトルになる。逆伝播の場合は、反対にサイズが $H \times W \times F$ のベクトルが、高さ H、幅 W、チャンネル数 F に変換される。

第 4 章　転移学習 [3)]

4.1　データが少ない場合の学習

深層ニューラルネットワークの学習は、一般に多くの訓練データを必要とする。しかし訓練データを望むままに用意できることはほとんどなく、少ないデータでいかに良い学習を行えるかが鍵になる。

あるタスク T を、小規模な訓練データ $D = \{(x_n, d_n)\}_{n=1,\cdots,N}$ のみを用いて効果的に学習するにはどうすれば良いかを考える。T にはここではクラス分類問題を主に考えるが、議論の大部分は他の問題でも有効である。また、本実験では転移学習を用いているため、転移学習について記述する。転移学習は、広い意味では別のタスク T' を学習して得た知識を T の学習に役立てることを指す。また、狭い意味では、特定の方法を指す。

4.2　転移学習の概要

あるタスクを解く際に獲得した知識を、別のタスクを解くための学習に転用することを転移学習 (transfer learning) という。その有効性の高さから、深層学習の中核的技術の一つに位置付けられている。本来、転移学習とは包括的な概念を表すが、深層学習でそれが指す方法はかなり明確である。目的とするタスク T と、それとは異なるものの一定の類似性を持つタスク T' があるとする。さらに T の訓練データ D は量が十分でない一方、T' のデータ D' は豊富にあるとする。ここでは T と T' それぞれの入力 (x と x') は同一の空間にあるが、出力 (y と y') の空間は違っていて構わないとする。この時転移学習とは、まずあるネットワークに D' を使って T' を学習させた後、このネットワークを若干改変したものに D を使って T を学習させる。具体的には次のように行う。まずネットワーク $Y' = f'(x')$ で D' を使って T' を学習する。その結果 f' は T' を十分うまくこなせるようになったとする。そんな f' を次のいずれかの方法で、T の学習に利用する。

- ネットワーク f' を T のための「特徴抽出器」として利用する。
- ネットワーク f' の構造と重みの一部をコピーして別のネットワーク f を作り、f で T を「再学習」する。なおこの方法はファインチューニングと呼ぶ。

これらの 2 つを以下で詳しく記述する。

なお、T から見た上述の T' の学習を、T の事前学習 (pretraining) と呼ぶ。逆に T' から見た T を、T' の下流タスク (downstream task) と呼ぶ。タスク T' と T は互いにある

程度近いものである必要があるが、その近さを定量的に測ったり、事前に転移学習の成否を予想するのは一般に困難である。

4.2.1 特徴抽出器としての利用

1つ目は、T' を学習した f' を T のための特徴抽出器として利用する方法である。f' の中間層を1つ選び、入力 x に対するこの層の出力 $z = z(x)$ を x の特徴とみなす。そして z を入力とする何らかの予想器（多くの場合サポートベクトルマシンや決定木などの基本的なものでよい）を訓練する。推論は入力 x に対する $z = z(x)$ を学習したい予測器に入れて行う。

この予測器の学習は次のように行う。目的タスク T の訓練データ $D = \{(x_n, d_n)\}_{n=1,\cdots,N}$ の各サンプルについて、入力 x_n を f' に入れた時の中間層の出力 $z_n = z(x_n)$ を計算し、新たなデータ $\tilde{D} = \{(z_n, d_n)\}_{n=1,\cdots,N}$ を得る。そして z_n から d_n を予測する予測器を学習する。取り出した特徴 z が T にとって良いものであれば、シンプルな予測器でも性能が出るはずである。予測器が単純である分、訓練データ $\tilde{D}$(と元となる D) の量が限られていても、過剰適合せずに学習が行えると期待される。

なお、z に f' のどの層の出力を選ぶかに自由度がある。1つの層だけでなく、複数の層の出力を組み合わせたものを z とすることもできる。どのようにするのが最良の結果につながるかは、様々な要因が関与し、最終的には実験的な検証に委ねられる。

4.2.2 ファインチューニング

もう1つの方法は、転移元のタスク T' を学習したネットワーク f' の一部コピーして新たにネットワーク f を作り、これを T の訓練データ D で訓練することである。T' と T が互いに近い場合、この訓練による f' から f への変化はわずかであることが想定されることからファインチューニング (fine-tuning) と呼ばれる。f の作り方に制約はないが、最も単純には f' の出力層を除く構造と重みをそっくりコピーする。一般には T と T' とで予測すべきもの (出力 y と y') が違うはずであるため、少なくとも出力層だけは作り変える必要がある。その際、新たに中間層を追加したり、または逆に削減したりしても構わない。

こうして作った f を、訓練データ D を用いて通常の方法で訓練する。ゼロから学習する場合との違いは、f に f' と同じ構造を持たせた上で、その重みの初期化をランダムに

行うのではなく、f' の重みで初期化することである。なおこの状態の f を、事前学習済み (pre-trained) であるという。新たに追加した層の重みは (他に方法がないため) ランダムに初期化する。

f の学習時には、そのすべての層の重みを更新しても良いし、一部の層の重みだけを更新することもできる。更新の対象とする層数が多いと、それだけ決定すべきパラメータが増えるため、一般に訓練データ D が大きくなければならない、したがって、与えられた D の量に応じて更新の対象とする層数を増減することが合理的である。つまり、D が豊富にあるときは、より高い予測性能を狙って多くの層を更新対象とする。逆に D が小さいときは、過剰適合を避けるため、重みを更新する層を絞る。もちろん、ランダムに重みを初期化した層は必ず更新しなければならない。

なお、f の全ての層ではなく一部の層を更新対象に選ぶ場合は、通常、出力層側から連続する層を選ぶ。これは多くの場合、f' および f では、入力に近い下位層ほど異なるタスクに共通する基礎的な (=「低次の」) 特徴を抽出する傾向があり、一方で出力に近い上位層ほどタスク依存度の高い「高次」の特徴を取り出すと考えられるからである。そのため個別タスクへの専門性がより強いと思われる上位層から順に優先して、更新対象に選ぶことが合理的である。

4.2.3 ImageNet 事前学習

画像分類を始めとする画像を扱う問題では、盛んに転移学習が活用されてきた。中でも ImageNet の物体カテゴリ認識は、様々なタスクに転移できることが知られている。畳み込みネットワーク (CNN) を用いて、ImageNet の 1000 種類の物体カテゴリを約 100 万枚の画像を使って学習する。この CNN がその内部に獲得する特徴表現は、画像を入力とする様々なタスクに対し、かなり普遍的に有効であることがわかっている。特徴抽出器としての利用、ファインチューニングのいずれの方法によっても、一般に訓練データ D が少ない場合でも、目的タスク T に対する推論を高い精度で実行できるようになる。これは T が ImageNet の物体認識に明らかに近い場合はもちろん、ネットワークの上部構造がそれほど似ていないように思われるタスク間でも当てはまることがある。

4.3 事前学習の有効性

D を用いて T を一から (=重みをランダム初期化した状態から) 学習するのではなく、T' を事前学習した状態から開始して学習することには、3 つの効用があると考えられている。すなわち、(a) 予測精度の向上、(b) 学習の収束速度の向上、そして (c) 推論時の頑健性の向上である。

このうち (a) の予測精度の向上は、われわれが事前学習に最も期待することであり、実際に多くの場合にはそれが観測される。タスク T' の学習を通じて獲得した特徴抽出の能力が、T で再利用されるためであると理解される。ただし、T の訓練データ D の量が十分である場合は、転移学習を行なっても性能は向上しないと考えられている。

(b) の学習速度の向上は、事前学習によって、より少ない重みの更新回数で同じ予測制度に達することを指しており、(a) とは違って D が大きい場合でもこの現象が見られる。事前学習が、ランダムな初期値よりも統計的に良い初期値を重みに与えているためであるという説が唱えられている。

(c) の推論の頑健性向上は、敵対的事例への防御、分布外入力の検出、さらに確信度の校正精度において、それぞれ性能が向上することを指す。これは事前学習によって、より高い表現力を持つ特徴空間が獲得されるためであると考えられる。事前学習なしに T を一から学習すると、T にとって必要最低限の特徴空間しか得られないため、それが差につながると考えられる。事前学習は、このように様々な効果を期待できる一方で、解明されていない部分も残されている。海外の文献では、入力 x の統計的な性質が大きく異なるタスク間の転移学習を分析し、特徴の再利用がほぼ生じないにもかかわらず、推論の精度が向上する場合があることが報告されている。また、この一見矛盾する現象は、ImageNet ようにデザインされた CNN が、目的タスク T に不釣り合いなほど規模が大きく、その負の影響を事前学習が緩和しているためであるとしている。

第5章　実験

5.1　実験方法

本実験では、転移学習とファインチューニングを用いて事前トレーニング済み畳み込みニューラルネットワークから猫と犬の画像を分類するという実験を行った。この際使用する事前トレーニング済み畳み込みニューラルネットワークは、Google が開発した MobileNet V2 モデルを使用する。

このモデルは、最初の畳み込み層 (Conv2D)、複数の反転残差ブロック (Inverted Residual Blocks)、最後の畳み込み層と分類器で構成されている。最初に 3×3 の畳み込み (Conv2D) を用いて特徴抽出を行い、その後 17 個の反転残差ブロックを通してデータの変換を行う。反転残差ブロックは、1×1 の Pointwise Convolution で特徴量の次元を拡張し、3×3 の Depthwise Convolution で空間的特徴を捉え、最後に 1×1 の線形畳み込みで特徴量の次元を圧縮するという 3 層構造を持つ。また、ストライドが 1 の場合は入力と出力を加算するショートカット接続を適用する。最後に、1×1 の畳み込み (Conv2D) で特徴量を統合し、Global Average Pooling によって空間次元を圧縮した後、全結合層と Softmax 層で最終的なクラスを分類するという構造となっている。この基本モデルを適宜変更して本実験に利用した。

変更内容としてはまず、転移学習をするために必要な変更を行う。モデルをコンパイルしてトレーニングする前に畳み込みベースを凍結させる。次に、この畳み込みベースの上に分類器を追加する。この分類器は、特徴ブロックから予測値を生成するために f.keras.layers.GlobalAveragePooling2D レイヤーを使用して 5×5 空間の空間位置を平均化し、特徴を画像ごとに単一の 1280 要素ベクトルに変換する。次に、tf.keras.layers.Dence レイヤーを適用して、これらの特徴を画像ごとに単一の予測値に変換する。この予測は logit または未加工の予測値として扱われるため、ここで活性化関数は必要ない。正の数はクラス 1 を予測し、負の数はクラス 0 を予測する。これによって、畳み込みベースが凍結され、トレーニング中に特定のレイヤーの重みは更新されなくなる。また、畳み込みベースを特徴抽出器として使用し、最上位の分類器のトレーニングを行う。ただし、ここでの上位レイヤーとは出力層付近のレイヤーのことを指す。

次にファインチューニングをするための変更を行う。転移学習の実験では MobileNet V2 基本モデル上に少数のレイヤーを重ねてトレーニングをしたに過ぎなかったため、事

前トレーニング済みネットワークの重みはトレーニング中に更新されない。パフォーマンスをさらに向上させる方法として、追加した分類器のトレーニングと並行して、事前トレーニング済みモデルの最上位レイヤーの重みをトレーニングする必要がある。ここで、大部分の畳み込みネットワークでは、上位のレイヤーに行くほど専門性が高くなる。最初の数レイヤーは、ほとんど全てのタイプの画像に一般化する非常に単純かつ一般的な特徴を学習する。上位レイヤーに行くに従って、特徴はモデルがトレーニングされたデータセットに対してもっと固有なものになっていく。ファインチューニングの目的は、一般的な学習を上書きするのではなく、特殊な特徴に適応させることであるため、少数の最上位レイヤーを解凍してトレーニングを行なっていく。具体的には、全 154 層あるうちの 100 レイヤー目から解凍してトレーニングを行う。

この転移学習の回数とファインチューニングの回数の比率を変更していき、どの比率の場合に一番効率よく学習を進めることができるかを検証していく。ただし、総学習回数は約 30 回とし、精度と損失を見て検証する。また、データは Google が提供している犬と猫のデータセット [4] を利用する。このデータセットは、訓練データが犬と猫それぞれ 1000 枚ずつ、検証データがそれぞれ 500 枚ずつあるデータセットである。

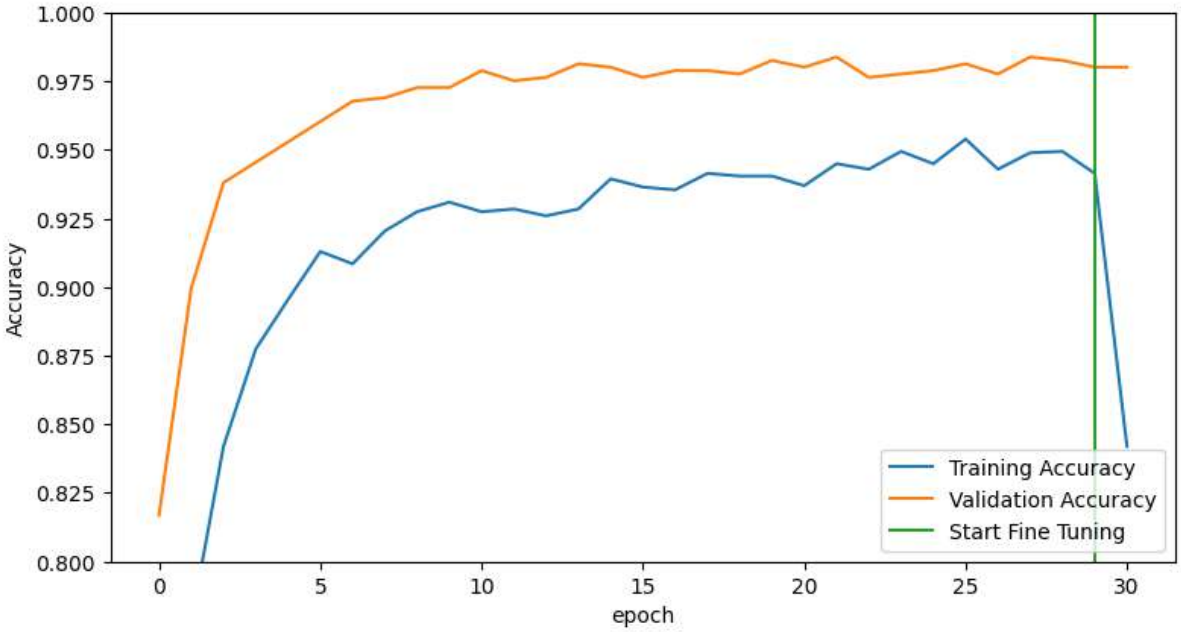
5.2 犬猫分類の実験結果と考察

転移学習 30 回、ファインチューニング 1 回の精度と損失のグラフを Figure 5.1、転移学習 20 回、ファインチューニング 11 回の精度と損失のグラフを Figure 5.2、転移学習 10 回、ファインチューンング 21 回の精度と損失のグラフを Figure 5.2、転移学習 0 回、ファインチューニング 31 回の精度と損失のグラフを Figure 5.4 にそれぞれ示す。これらのグラフを見ると、ファインチューニングを行った瞬間にトレーニングデータの精度、損失ともに一度大幅に結果が悪くなっていることがわかる。しかし、この現象はトレーニングデータのみで検証データではこのような現象は起きていない。また、全てのグラフを見比べても、ファインチューニングの回数が多いグラフの方が精度、損失共に高いパフォーマンスを出すための学習回数が少ないことがわかる。このことから、ファインチューニングを早い段階で始めた方が精度と損失の収束が早く、高精度な結果になるということがわかる。

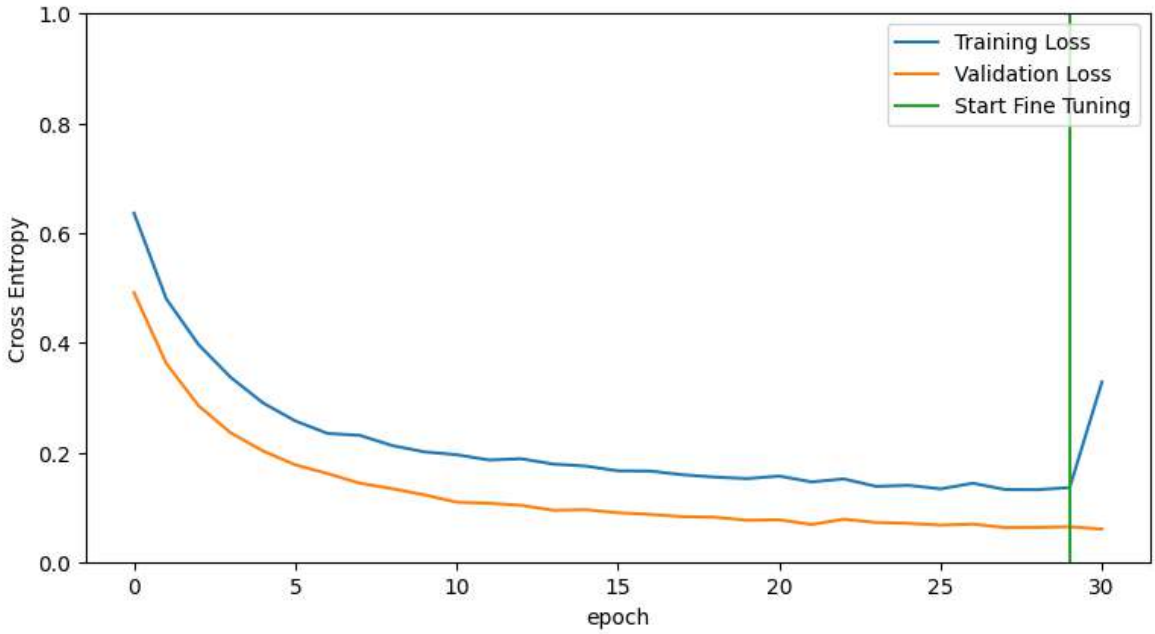
検証データでは、各方法で収束した損失の値にも違いがみられた。Figure 5.1 の方法では、損失約 0.07 に収束したが、Figure 5.4 の方法では約 0.04 に収束した。また、トレー

ニングデータでは各方法で収束した精度と損失両方の値に違いがみられた。Figure 5.1 の方法では精度は約 94%、損失は約 0.14 に収束したが、Figure 5.4 では精度約 98%、損失約 0.05 に収束した。これらのことからもファインチューニングの回数が多い方が結果が良くなることがわかる。

ファインチューニングを行った瞬間にトレーニングデータの精度、損失ともに一度大幅に結果が悪くなる現象について考察する。これは、ファインチューニングを開始するときに今まで凍結していたモデルの最上位レイヤーを急に解凍するため、そのレイヤーがデータに急に対応できず結果が悪くなってしまっていると考えられる。また、トレーニングデータは 1epoch 中の平均値であるため、学習はじめは結果が一時的に悪化する。しかし、検証する際には元通りになっているため、検証データには影響しないということも考えられる。

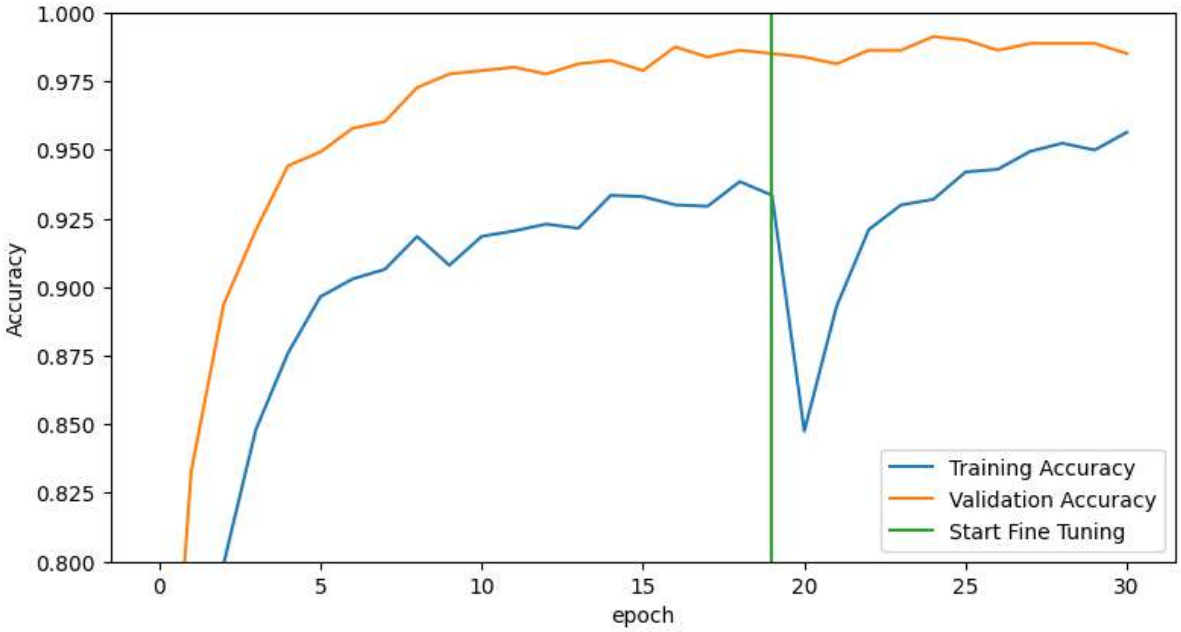


(a)Training on Accuracand Validatiy.

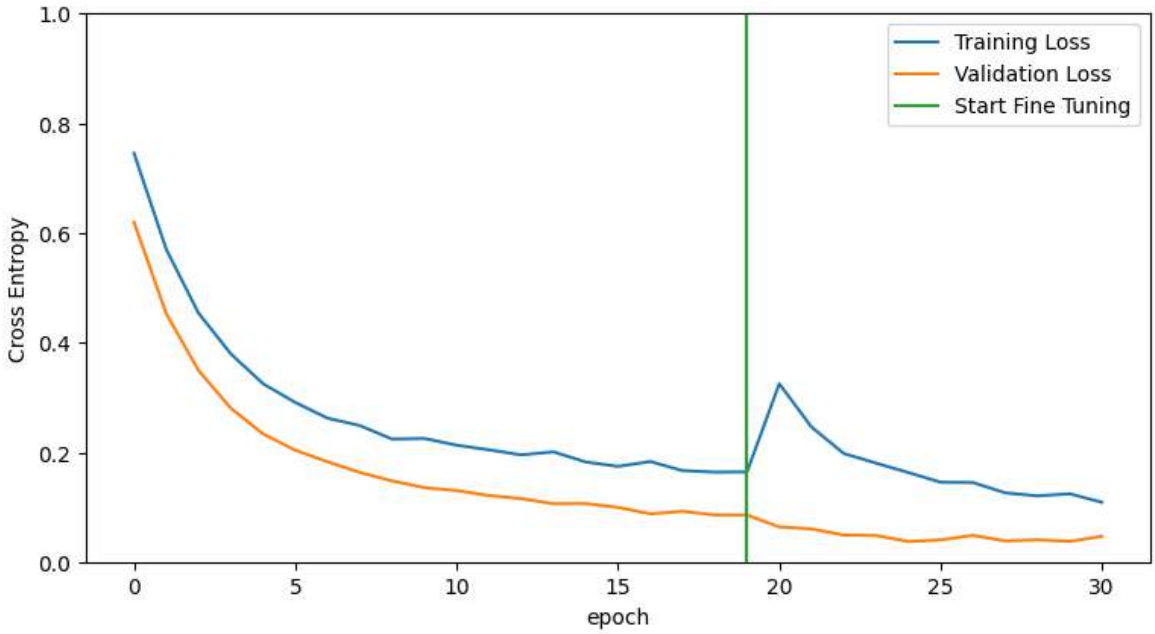


(b)Training and Validation Loss.

Figure 5.1 30 epochs for transfer learning, before 1 epoch for fine-tuning.

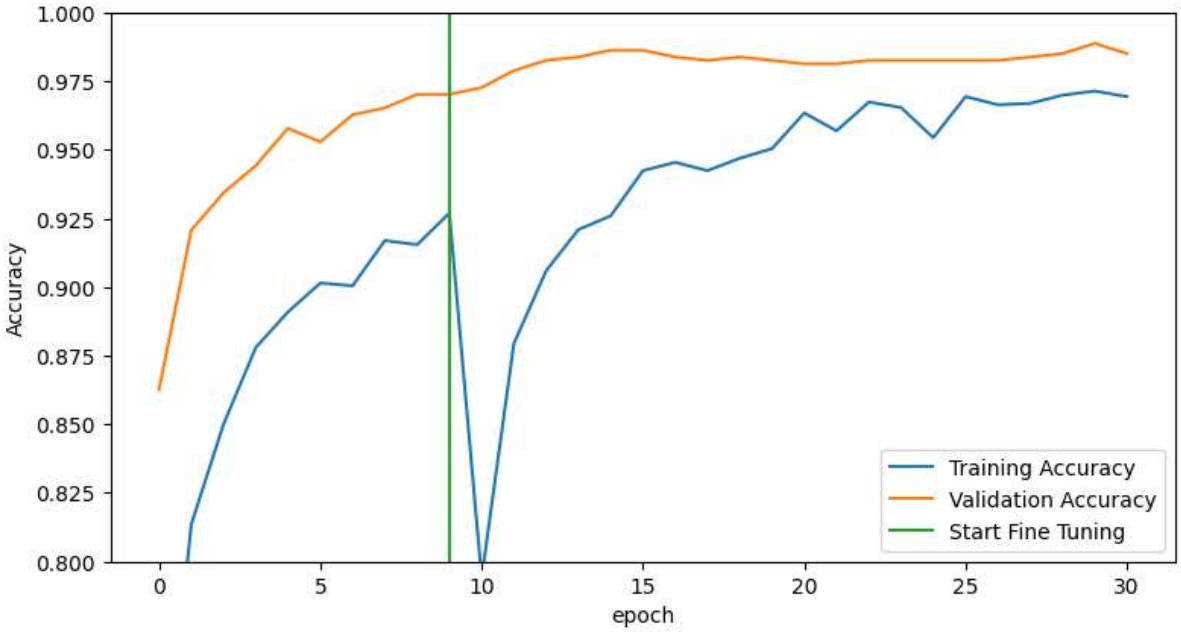


(a)Training on Accuracand Validatiy.

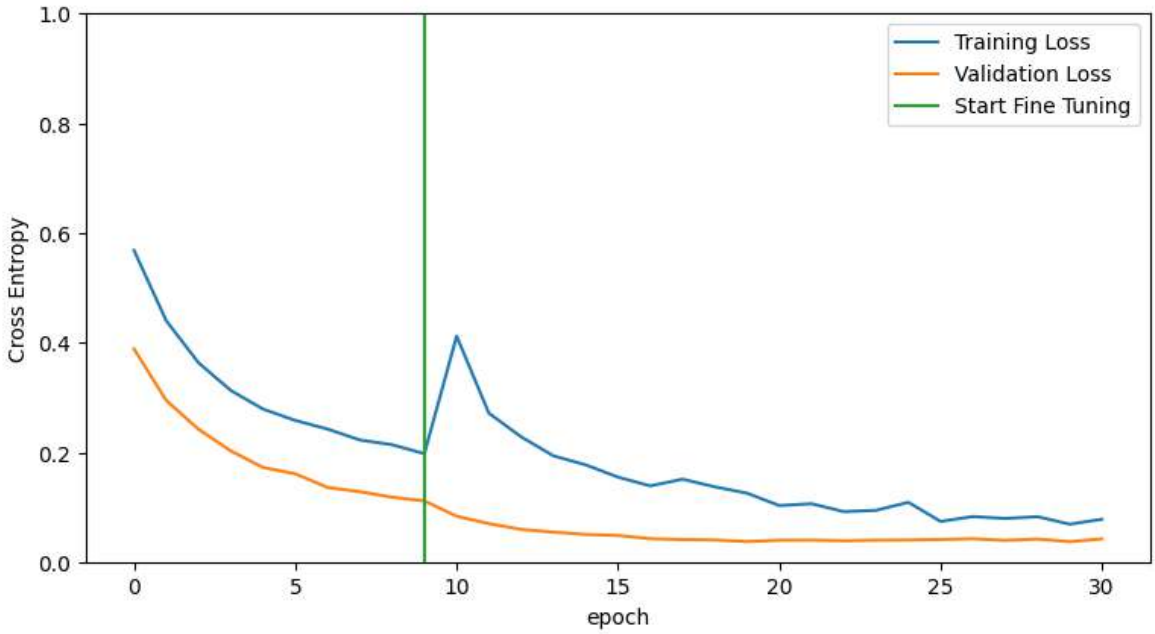


(b)Training and Validation Loss.

Figure 5.2 20 epochs for transfer learning, before 11 epoch for fine-tuning.

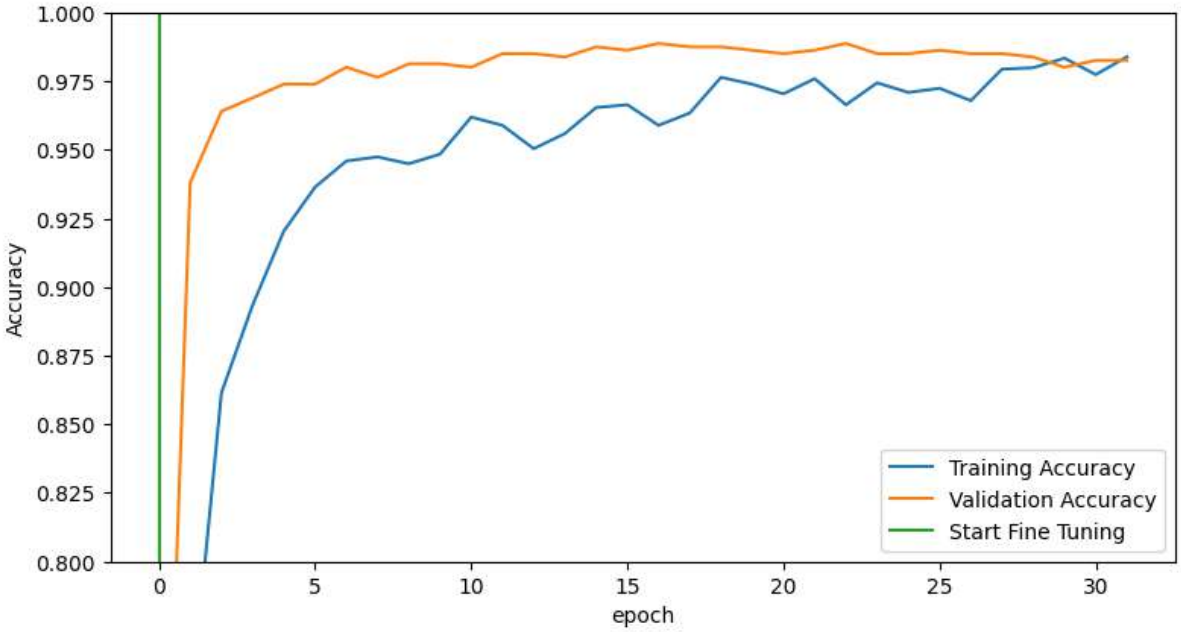


(a)Training on Accuracand Validatiy.

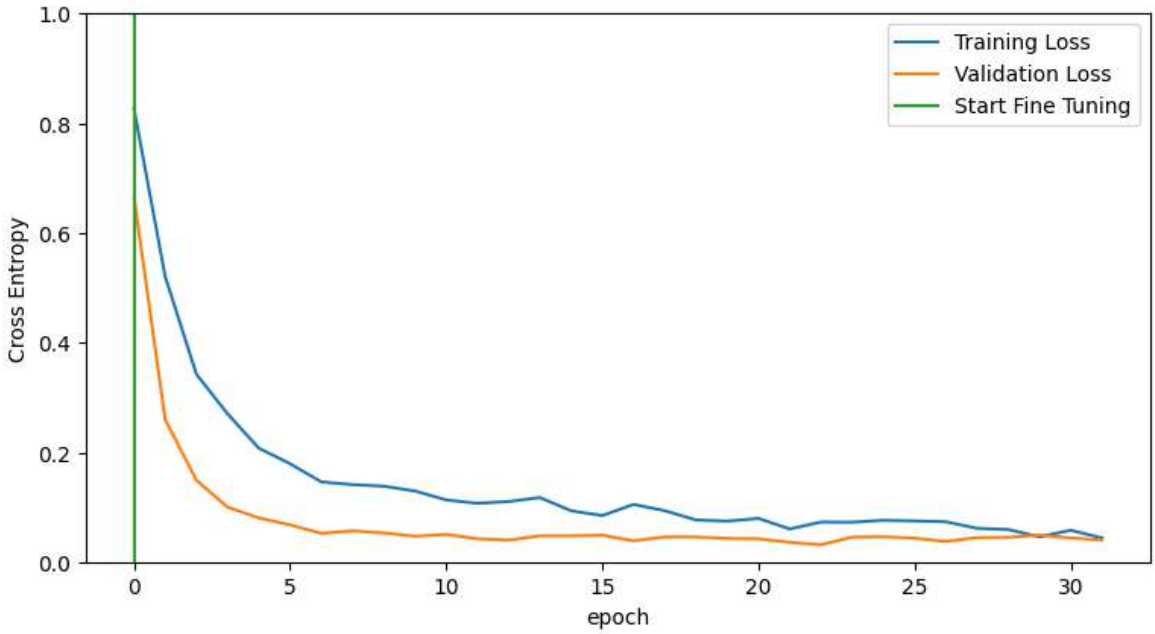


(b)Training and Validation Loss.

Figure 5.3 10 epochs for transfer learning, before 21 epoch for fine-tuning.



(a)Training on Accuracand Validatiy.



(b)Training and Validation Loss.

Figure 5.4 0 epochs for transfer learning, before 32 epoch for fine-tuning.

5.3　バラとチューリップの分類実験と考察

上述の犬猫分類実験ではグラフを見てもわかるように、全体的にかなり少ない学習回数で結果が収束していることがわかる。これは、犬と猫の分類が簡単であることや、データが簡単であることも要因として考えられる。そのため、バラとチューリップの画像のデータセット[5)]を用意しもう一度実験を行うことにする。ただし、データセットの中身を確認したところかなり複雑なデータセットであった。具体的には、風景の中に小さいチューリップがあるだけの写真や、絵画がメインの写真で額縁にバラがついている写真などが確認できた。データ数としては、バラが 641 枚、チューリップが 799 枚あり、そのうちの 80%をトレーニングデータ、20%を検証データとした。また、犬と猫の分類よりもバラとチューリップの分類のほうが難易度が高いと考えられるため、総学習回数は約 50 回として実験を行う。学習回数の比率は、犬猫分類実験と同じように転移学習 50 回とファインチューニング 1 回、転移学習 40 回、ファインチューニング 11 回 · · · というように変化させていく。

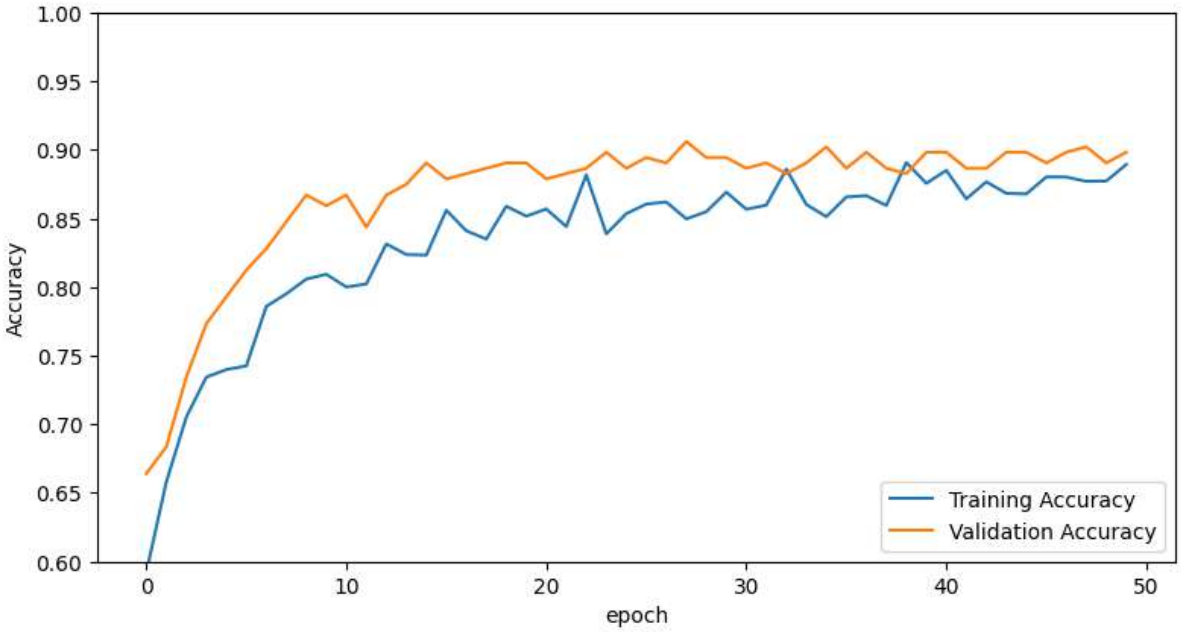
学習回数の比率を変化させていった中で、一番結果が良かったものと一番結果が悪かったものについて述べていく。一番結果が良かった比率は、転移学習 0 回、ファインチューニング 51 回であり、一番結果が悪かったものは転移学習 50 回、ファインチューニング 1 回であった。それぞれの結果を Figure 5.6、Figure 5.5 に示す。ただし、Figure 5.5 の精度のグラフは 0.60 から 1.00 の範囲で、Figure 5.6 の精度のグラフは 0.80 から 1.00 の範囲で描写している。

これらの結果を詳しく見ると、Figure 5.5 の検証データの精度は約 89%、損失は約 0.26 が最大となった。また、精度の結果が収束するのは学習回数約 20 回ほどのタイミングで、損失は約 35 回ほどのタイミングで収束した。そして、犬猫の分類実験と比較すると、検証データのグラフとトレーニングデータのグラフとの差が小さくなっていることがわかる。Figure 5.6 の検証データの精度は約 93%、損失は約 0.15 が最大となった。そして、精度の結果が収束するのは学習回数約 45 回ほどのタイミングで、損失は約 30 回ほどのタイミングで収束した。Figure 5.5 と比較すると収束するタイミングは遅くなっているが、Figure 5.5 の最良の結果になるまでの時間は Figure 5.6 のほうが早いという結果となった。

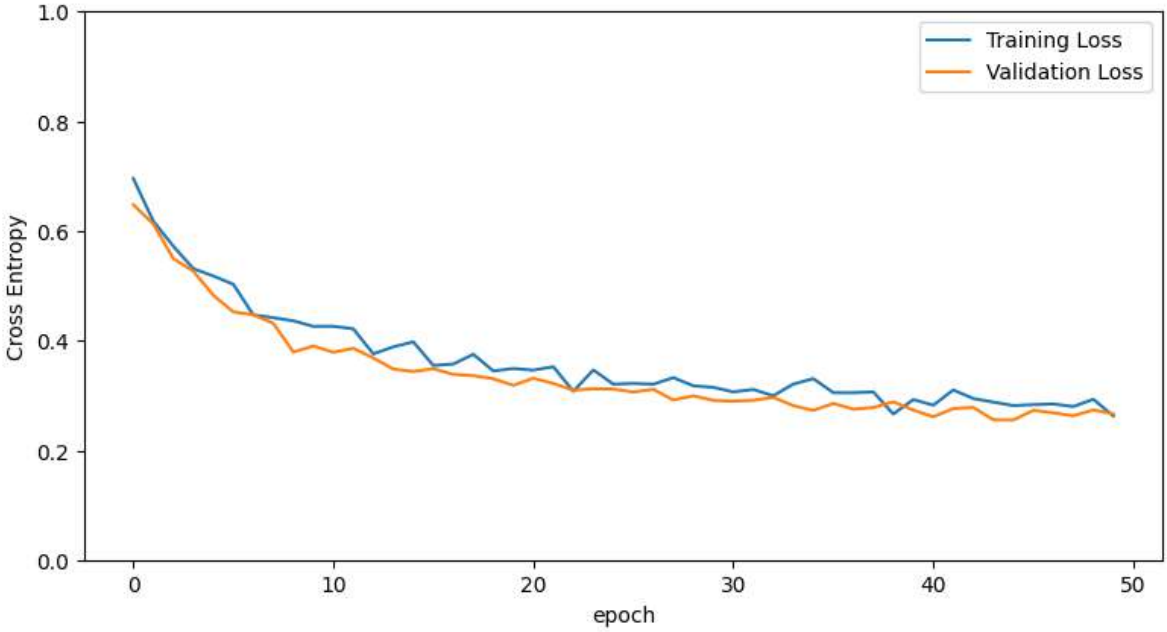
この結果のみを見ると Figure 5.6 のほうが良い結果となったといえるが、Figure 5.6 は検証データの結果に対してトレーニングデータの結果が異様に良いことがわかる。これ

は、トレーニングデータの画像に対応しすぎた故に、検証データに対応できていないという過学習に似たような現象が起きていると考えられる。そのため、Figure 5.6 のほうが良い結果であるとは一概には言えないという結果となった。

ファインチューニングの回数を増やすと過学習が起きた原因について考察する。これは、ファインチューニングは、転移学習よりも重みを更新するレイヤーの数が多いためであると考えられる。そのため、転移学習よりもトレーニング中に多くのレイヤーが更新されていく。これによって、トレーニングデータに過剰に適合してしまい、過学習が起きたと考えられる。

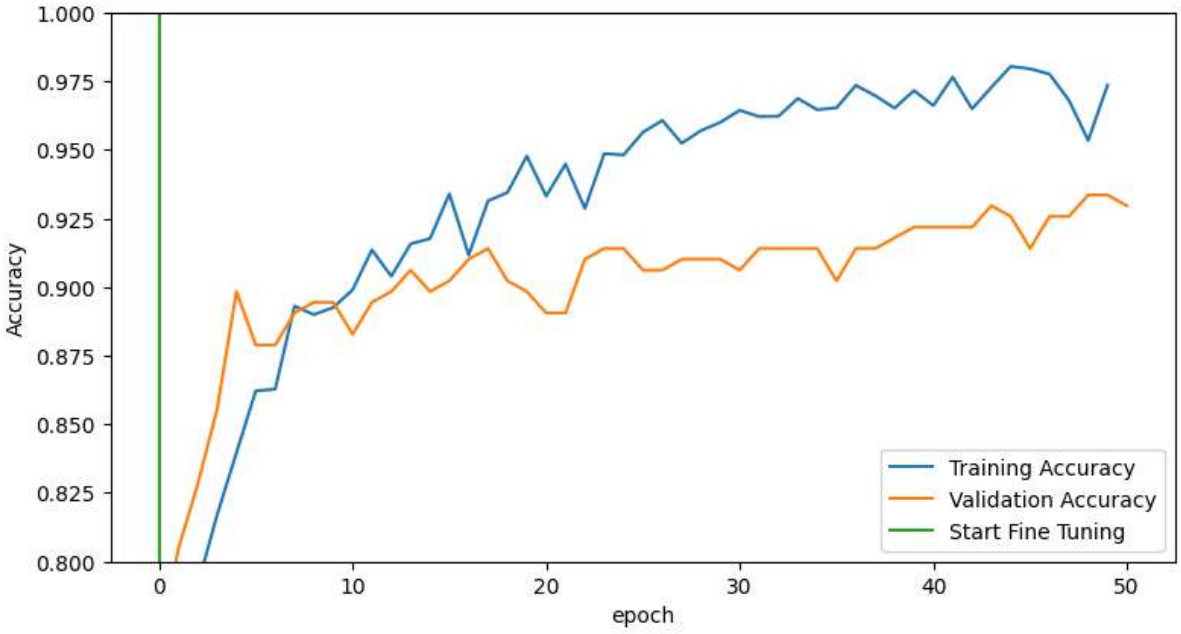


(a)Training on Accuracand Validatiy.

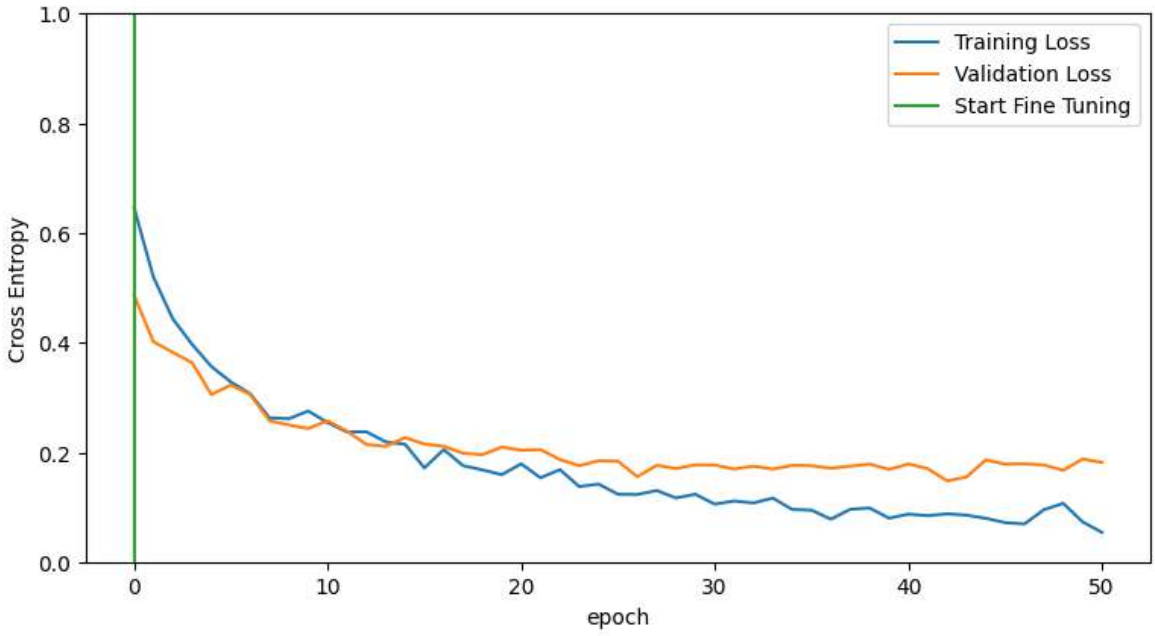


(b)Training and Validation Loss.

Figure 5.5 50 epochs for transfer learning, before 0 epoch for fine-tuning.



(a)Training on Accuracand Validatiy.



(b)Training and Validation Loss.

Figure 5.6 0 epochs for transfer learning, before 51 epoch for fine-tuning.

第6章　結論

本研究では、転移学習とファインチューニングの回数の比率による学習効率の変化について研究を行った。本研究の一つの目標としては、転移学習とファインチューニングの両方を利用して、通常の学習よりも効率の良い学習を行うことであった。しかし、結果としてはファインチューニングの比率を高くするほど効率の良い学習ができるという結果となった。

考えられる要因としては、今回使用した事前トレーニング済み畳み込みニューラルネットワークであるMobileNet V2が既に良いパフォーマンスを出しているからというものが考えられる。ファインチューニングの目的としては、全体的な学習が終了したのちに微調整のために行われることが多い。そのため、既に良いパフォーマンスが出ているMobileNet V2モデルの微調整を早い段階で行うことによって、効率の良い学習ができたと考えられる。しかし、バラとチューリップの分類実験から、ファインチューニングの回数を増やしすぎると過学習が起きやすくなるという結果が出た。そのため、効率の良い学習を目指すには、学習回数を減らすということも視野に入れておく必要がある。

本実験では事前学習済みモデルはMobileNet V2のみを使用したためこのような結果となったが、モデルを変更すると結果は変わる可能性は大いに考えられる。そのため、結論としては、MobileNet V2モデルを利用して転移学習を行う場合、ファインチューニングを早めに開始し、回数は少なめにする。こうすることによって、過学習を起こす確率を減らし、効率の良い学習ができるという結論となった。

参考文献

1) 庄司雄太, 張力峰:CNN を用いた特定魚種の識別精度向上のための研究,2019 年度電気・情報関係学会九州支部連合大会（第 72 回連合大会）講演論文集,10-2A-07
2) 我妻幸長, 初めてのディープラーニング Python で学ぶニューラルネットワークとバックプロパゲーション ,SB クリクリエイティブ株式会社,2019
3) 岡谷貴之, 深層学習, 講談社,2022
4) https://storage.googleapis.com/mledu-datasets/cats_and_dogs_filtered.zip
5) https://storage.googleapis.com/download.tensorflow.org/example_images/flower_photos.tgz

謝辞

本研究を進めるにあたり、出口利憲教授に基礎知識など様々な助言をいただき、深く感謝いたします。