university of groningen / faculty of mathematics and natural sciences / artificial intelligence

University of Groningen,
the Netherlands

Master's Thesis
Artificial Intelligence

Hedge Backpropagation in Convolutional Neural Networks

Floris van Beers
s2197367

Internal Supervisor:
Dr. M. A. Wiering (Artificial Intelligence, University of Groningen)
[Dr. H. Jaeger (Artificial Intelligence, University of Groningen)]
External Supervisors:
Dr. L. Hogeweg (Machine Learning Researcher, Intel Corporation)

November 24, 2021

Abstract

In the field of object recognition much is gained by the use of convolutional neural networks (CNNs). Research into deeper networks has revealed untold successes, as well as unforeseen issues. Ensembles of deep CNNs are being used to further improve performance, while the issues with vanishing gradients have been tackled by the use of deep supervision. In this work a novel architecture is proposed which combines these techniques. Using ResNet34 and DenseNet121 as base variants, a Multiple Heads (MH) adaptation attempts to improve performance and solve issues. Further work on the weights (α) in the MH variant leads to use of the Hedge Back Propagation (HBP) algorithm in the HBP and Thaw model variants. Experiments on CIFAR10 and the Naturalis Papilionidae datasets show the use of MH variants improves over base networks in one of the experimental settings. The application of HBP does not further improve the performance of the MH variant, but leads to interesting observations resulting in a multitude of directions for future work.

Acknowledgement

For being my biggest supporter during my studies, my internship and my thesis, I want to thank my wife, Nury, from the bottom of my heart.

It is with great sadness that I say goodbye to Marco Wiering, who's guidance helped start, shape and grow this project. He has been a mentor to me throughout this and other projects, and my studies as a whole. His sudden passing came as a shock and has prohibited us from seeing the project through together. His contributions to our shared projects, and to the science community as a whole, will not be forgotten.

I am indebted to Herbert Jaeger, who stepped up on very short notice to finish the project with me. I am thankful for his wise words on the project, as well as Marco's passing.

To be able to work on my research during an internship at Intel Corporation, to meet many experienced, driven and wonderful people, I am grateful. I want to explicitly state my appreciation for all the time, energy and ideas Laurens Hogeweg invested in me as my project supervisor.

For all the support during my years of study, personal, financial and educational, I thank my loving parents.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Image classification is a task that has seen numerous solutions, with differing, but increasing, amounts of success. The number of applications is equally large, and growing. Due to all this interest in image classification, improvements have been made at a rapid pace. The task can be defined as using a system of rules to automatically label an image with the correct class label in a specific set of images. From this definition it can be seen that attempts to solve image classification can be performed simply by hand, by flow-chart, by more complex mathematical analysis, and ultimately by automated pattern recognition.

The biggest breakthrough in image classification in the last decades has been with the introduction of the convolutional neural network (CNN) [Lecun et al., 1998] and its first show of force [Krizhevsky et al., 2012]. CNNs have been the go-to system of rules applied to image classification. While this narrows the range of solutions to the task, these models have shown to be very effective. Additionally, this choice of solution in no way narrows the immensely broad applications for this task. These applications include object classification by cameras on an autonomous robot for scene understanding [Ye et al., 2017], medical imaging to determine the health of a tissue [Gertych et al., 2019], fine-grained animal classification to aid in determining biodiversity [Marre et al., 2020], and so on.

With the success of AlexNet [Krizhevsky et al., 2012] came the increased attention by the scientific community that led to numerous improvements to the first functional CNN. While many of these improvements were developed to incrementally increase the effectiveness of a model, some improvements were more drastic, both in structural impact as well as necessity. Most notable has been the effort to train ever deeper CNNs which introduced the vanishing gradient problem.

The vanishing gradient problem occurs when CNNs are made consistently deeper, meaning they consist of more layers. In general terms, the problem occurs when models are so deep that any information used to train the network is lost during backpropagation. Due to this, the earlier layers of a network do not receive any information from which they can learn. The solution to this problem has seen many forms, from mathematical adjustments such as a different activation function [Xu et al., 2015], to structural solutions such as adjusting the connections between shallow and deep layers [He et al., 2016, Huang et al., 2017b]. The most impactful structural change to accommodate for this problem has been the use of auxiliary heads [Szegedy et al., 2015], a technique more generally known as deep supervision [Wang et al., 2015, Lee et al., 2015]. The addition of auxiliary heads allows shallow layers to be trained more directly, allowing more information to be learned by these layers. It has shown to be effective in training deep neural networks that would otherwise be untrainable.

A second, large, structural change used to improve results beyond incremental advances has been ensemble learning. This technique uses a group, or ensemble, of effective models and combines their output to produce its final prediction [Dietterich, 2000]. Using multiple models trained on similar data can produce an ensemble where each individual model's bias towards local minima or specific over-represented classes is negated by the combined output of the group. However, an ensemble in the naive sense requires training multiple models which, obviously, requires more time and computing power. To reduce these negative effects of using an ensemble, several methods try to reuse parts of the model in the ensemble [Minetto et al., 2019, Huang et al., 2017a].

In this work, several novel model structures and a corresponding training algorithm are introduced, adapted from work in online deep learning [Sahoo et al., 2018]. These novel structures use the auxiliary heads as seen in deep supervision [Szegedy et al., 2015, Wang et al., 2015, Lee et al., 2015]. In contrast to these techniques, the multiple heads (MH) are kept active during inference, resulting in a model that is an ensemble of itself and shallower versions of itself. In addition to this new model structure, the technique of hedge backpropagation (HBP) [Sahoo et al., 2018] is adapted to convolutional neural networks. This technique then allows the model to learn the weight of each head during training, resulting in two additional model structures. The models using MH without HBP are named MHResNet34 and MHDenseNet121, based on ResNet34 and DenseNet121 respectively. The models using HBP are named HBPResNet34 and HBPDenseNet121. Finally, a hybrid implementation of MH and HBP results in ThawResNet34 and ThawDenseNet121.

1.1 Research Questions

The development of the model structures described in Chapter 1 and their effectiveness compared to the base models is the core of this work. This comparison can be made in several steps, each represented by a separate research question:

1. Does a model with auxiliary heads perform better than the base model?

2. Do the auxiliary heads increase performance when active during inference, in contrast to only using auxiliary heads during training?

3. Does the hedge backpropagation (HBP) algorithm successfully optimize weights between output layers of a model with multiple heads?

4. Does freezing the weights assigned to classifiers in the model at the start of training influence the performance of the model? Does this significantly differ from the MH or HBP models?

5. Do the behaviours of these models change when using differing amounts of data?

1.2 Thesis Structure

This section has given a very brief summary of the goals of this work. In Chapter 2, each topic mentioned in this summary will be more fully put into its scientific context. Chapter 3 will then introduce the new additions of this work formally, together with the data used. To measure the effectiveness of these methods, a multitude of experiments was set up, described in Chapter 4. The experimental results are shown in Chapter 5, together with the statistical analysis. Finally, Chapter 6 will answer the research questions, along with other relevant conclusions, before listing suggestions for future research.

Chapter 2

Theoretical background

2.1 Deep Learning

The origins of Deep Learning (DL) can be traced back all the way to the Rosenblatt perceptron [Rosenblatt, 1958]. This mathematical operation is nothing more than a weighted sum of inputs, producing an output of 1 if this sum crosses a threshold and 0 otherwise. This operation, in addition to a function to update the weights, is the basis of the artificial neuron. Several adaptations and extensions made to the perceptron have produced the earliest neural network models called Multi-Layer Perceptrons (MLPs). These adaptations are stacking the perceptrons in layers, using a more complex update algorithm, and using non-linear activation of the individual neurons.

2.1.1 Neural Networks

The effectiveness of neural networks lies in its power to update weights of multiple layers of neurons with non-linear activations by determining each neurons contribution to the output. The usefulness of these models is in the use of non-linear activations. By stacking non-linear activations, a neural network as simple as an MLP can be used as an universal function approximator.

To train a neural network for a specific task an iterative procedure is applied of alternating between forward and backward passes. The forward passes feed the network, or model, a set of inputs to which the network then connects an output, e.g. a class label. The backward passes calculate the error between the output and the desired label and update the weights of each layer according to this error in a process called backpropagation [Rumelhart et al., 1986]. Finally, the iterative process of updating the weights in the direction that minimizes error is done by a gradient descent algorithm. The necessity for labeled samples makes training a neural network a supervised learning task.

Backpropagation

The technique of backpropagation [Rumelhart et al., 1986], which allows networks with more than a single layer to train the weights connected to non-output layers, is a cornerstone of neural networks. The use of chained partial derivatives allows information about the gradient used by gradient descent to flow backward through the model, starting at the output layer. This allows a single error calculation in the final layer to affect all weights in the model.

2.1.2 Convolutional Layers

A big leap forward in using neural networks for image processing has been made through the development of Convolutional Neural Networks [Krizhevsky et al., 2012, Simonyan and Zisserman, 2015]. These networks, as the name suggests, use convolutions, built into seperate layers called convolutional layers. These layers use feature maps to detect features in localized areas of their input, giving them the possibility of detecting spatial relations between input values (pixels). This allows much more relevant information to be extracted from images. Additionally, applying filters, or kernels, to these localized areas allows for weight sharing, since a single kernel has identical weights regardless of which patch of the input it is currently applying the filter to. These two factors allow for more information retention with less parameters, greatly increasing performance on image-based data, where spatial relations are relevant.

Multiple stacked convolutional layers can turn lower level features, such as lines and shapes, into higher level features such as facial features, windows in a building, or legs of an animal. Together with intermittent pooling layers, these features are effectively combined to correctly identify an entire object by adding numerous features together into a single output. The final output of features is then used by a conventional (set of) layer(s) to correctly label the input.

2.1.3 Vanishing Gradient Problem and Solutions

Initial attempts to improve CNNs frequently led to increases in the number of convolutional layers, with 2 layer in LeNet [Lecun et al., 1998], 5 layers in AlexNet [Krizhevsky et al., 2012], and 16 in VGG19 [Simonyan and Zisserman, 2015]. Attempts to go larger ran into issues with backpropagation. This issue was coined the vanishing gradient problem [Hochreiter, 1991, Kolen and Kremer, 2001].

The vanishing gradient problem describes the inability for significantly deep neural networks to learn any significant patterns in their early layers. This is caused by the process of chaining partial derivatives. If we assume the activation function of each layer has gradient $< 0, 1 >$, such as with the sigmoid activation (Equation 2.1) or hyperbolic tangent activation (Equation 2.2), the gradient g_i of layer i will be $g_i =< 0, 1 >^i$ due to the chained nature of backpropagation. The more these gradients approach 0, the quicker the value for g_i approaches 0. A layer with gradient 0 will not update its weights and therefore not learn anything. If this happens later in the learning process, the route to an optimal solution is stifled. However, if this happens much earlier, the early layers will be untrained and will be, at best, useless, and might even be detrimental by acting as a filter applying noise to the input.

$$\sigma(x) = \frac{1}{1 + e^x} \tag{2.1}$$

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.2}$$

An initial, and intuitive, response to this problem is to replace the activation function with one that does not exhibit this behaviour. The current standard for this is the rectified linear unit (ReLU) activation. This activation, described by Equation 2.3, provides a gradient that is either 1 or 0, removing the situation of arbitrarily small gradients being multiplied and vanishing.

$$ReLU(x) = max(0, x) \tag{2.3}$$

Another solution is to increase the connection between the output layer and early convolutional layers in the network. These connections are generally called skip connections, as they skip one or more layers of the model. These skip connections ensure information from earlier layers reaches the final output and thus impacts the gradient for these early layers, negating the devastating effect of the vanishing gradient problem. Skip connections come in two variants: additive connections, leading to ResNet variants [He et al., 2016], where the input of a block of layers is added to the output of that block, and concatenated connections, resulting in DenseNet variants [Huang et al., 2017b], where the features of previous blocks are added to the output of the current block as a combined input to the next block. Both these variants and their workings are further explained in Section 3.2.

Regardless of technique, the goal of solving the vanishing gradient problem is to allow the training of deeper neural networks. Another technique developed for this goal structurally changes the architecture of the model by adding additional output layers. This area of research is known as deep supervision.

2.2 Deep Supervision

Deep supervision is the technique of adding auxiliary output branches, or auxiliary heads, to the model at halfway points to give earlier layers more direct feedback [Wang et al., 2015, Lee et al., 2015]. Extending the concept of supervised learning, this allows the supervisor (i.e. the user with labeled data), to provide this feedback at multiple depths.

Deep supervision has been used to train state-of-the-art models, such as Inception [Szegedy et al., 2015] and Inception v3 [Szegedy et al., 2016]. These models use auxiliary classifiers at predetermined points during training. At inference time, when the model is sufficiently trained at every layer, the auxiliary heads are disconnected, producing a feed forward network. During training, the auxiliary heads provide additional gradient information according to preset weights, which are hyper-parameters to be optimized by the user.

Deep supervision has seen many developments, such as using intermediate concepts [Li et al., 2019] and knowledge synergy [Sun et al., 2019].

With the wide use of certain popular models such as ResNet50 [He et al., 2016] and Inception V3 [Szegedy et al., 2016], deep supervision has fallen off in use due to the prevalence of pretrained weights. However, it is also shown that when models are developed specifically for new tasks, and as such no pre-training is available, use of deep supervision can greatly improve the training time and performance of these models [Shen et al., 2020], or might even be the factor that allows training altogether [Szegedy et al., 2016].

2.3 Ensemble Learning

While new models are continually developed and improve on the state-of-the-art, there is no guarantee that the best performing model overall, in terms of accuracy, performs the best in every possible case of that task. In other words, two models of comparable architectures might be specialized in detecting two different classes, which performing worse in the class the other model is specialized in. To use this information as a benefit, ensemble learning has been developed [Dietterich, 2000, Zhou et al., 2002].

While there are many ways to combine the used models into an ensemble, the baseline remains that multiple models need to be trained for the ensemble to be effective. Naturally, this puts a heavier burden on computational resources compared to training a single model. To this end, architectures have been developed that reuse some of the weights of the model, to reduce the necessary resources. This has been done by weight sharing up to a certain depth and fine-tuning only the final layers of the same model in different directions [Minetto et al., 2019]. Alternatively, models have been trained to convergence, while intermediate weights are saved as snapshots. These snapshots are then used to initialize separate instances of the same model, which can then be combined into an ensemble [Huang et al., 2017a]. These architectures show the necesity for ensemble learning to be more efficient than the naive approach of training many different models individually.

Chapter 3

Methodology

While deep supervision and ensemble learning are in itself established developments, this chapter will introduce a novel method, intended to combine strengths from both these concepts.

3.1 Hedge Algorithm

The hedge algorithm [Freund and Schapire, 1997] is an algorithm developed to determine the proper weights to assign to each output in a system of experts in order to obtain the best aggregated performance. The algorithm, as shown in pseudo-code in Algorithm 1, can be used for many purposes.

Algorithm 1: Hedge Algorithm Weight (α) Updates

```
Data: Pair of (data, label) = (x,y)
Result: Adapted weights $\alpha$
$\alpha_i(0) = \frac{1}{N}$ where $N$ is number of experts;
t = 0;
T = number of epochs;
for $t$ in range(T) do
    for $i$ in range(N) do
        $\hat{y}_i = expert_i(x)$;
        $l_i(t) = f(\hat{y}_i, y)$;
        $\alpha_i(t+1) = \alpha_i(t) \times \beta^{l_i(t)}$;
    end
end
```

In this algorithm, α is the array of values α_i, where i indicates a single Expert. $f(\hat{y}_i, y)$ denotes the loss function applied to prediction $\hat{y}_i$ and true label y. Finally, β is the discount factor, between 0 and 1, used to determine the reduction in weight α_i as a result of the loss produced by expert i. The value for β is set in the range $0 < \beta < 1$. A β value of $<= 0$ would put the weight for that expert at or below 0, making it irrelevant. In contrast, a β value of $>= 1$

would mean the weight is increasing with a higher loss value, giving the opposite of the desired effect. In practice, however, the β value is a value close to 1, such as 0.99, to ensure a gradual change in α.

In any environment with multiple experts with uncertainty to which experts performs best, the hedge algorithm can be used to determine expert comparative effectiveness. The algorithm has been used, for example, in stock trading [Yutong and Zhao, 2015], and disease classification [Sigcha et al., 2021]. The algorithm is specifically designed for such a system of experts to converge on a result approximating the result of the best expert. It can be proven that the difference in loss between the total model and the best expert decreases at the rate of $O\sqrt{(\ln N)/T}$, as done so in the work where the Hedge algorithm was introduced [Freund and Schapire, 1997]. As such, given enough time, the hedge algorithm will perform equal to a best expert, without needing the knowledge of which expert is best beforehand.

3.1.1 Hedge Backpropagation

An ensemble of neural networks is effectively a system of experts with uncertainty about the best performing expert. Multiple methods exist of combining model outputs into a single system output, such as averaging or voting, a weighted combination is the most adaptive method. To determine these weights, the hedge algorithm can be used. Hedging produces an extra weight, or α_i for each model i to be used at inference to determine the output of the system. However, this α_i can also be used in updating the model, resulting in an additional factor in determining learning speed, shown in Equation 3.2.

$$\theta(t+1) = \theta(t) - \mu\frac{\delta l(t)}{\delta\theta} \tag{3.1}$$

$$\theta_i(t+1) = \theta_i(t) - \alpha_i\mu\frac{\delta l_i(t)}{\delta\theta_i} \tag{3.2}$$

In Equation 3.2, compared to Equation 3.1, the addition of α_i and the specification of other network specific variables shows that this algorithm can be used to update a specific network in the ensemble. In order to ensure the functionality of the ensemble, certain concessions have been made deviating from the pure implementation of the hedge algorithm shown in Algorithm 1. The additional steps can be seen in Algorithm 2.

Algorithm 2: Hedge Back Propagation Algorithm Weight (α) Updates

```
Data: Pair of (data, label) = (x,y)
Result: Adapted weights $\alpha$
$\alpha_i(0) = \frac{1}{N}$ where $N$ is number of experts;
t = 0;
T = number of epochs;
for $t$ in range(T) do
    for $i$ in range(N) do
        $\hat{y}_i = expert_i(x)$;
        $l_i(t) = f(\hat{y}_i, y)$;
        $\alpha_i(t+1) = \alpha_i(t) \times \beta^{l_i(t)}$;
        $\alpha_i(t+1) = \max(\alpha_i(t+1), \frac{s}{N})$
    end
    $\alpha = \frac{\alpha}{\mathrm{sum}(\alpha)}$
end
```

The additional steps in the hedge backpropagation (HBP) algorithm provide the α updates with two important properties. First, the α_i of a specific expert i is lower-bound to a predetermined value s, divided by number of experts N. Second, the array of weights α is normalized at each iteration, providing any individual α_i with the upper bound shown in Equation 3.3

$$1 - (\frac{s}{N} \times (N-1)) \tag{3.3}$$

The application of HBP is intuitive, but its usefulness is not. Simply put, the HBP algorithm allows the best performing network in the ensemble to train quicker and perform better, while worse performing networks will learn slower. This causes the discrepancy between best and worst network to become larger. When using completely individual networks in an ensemble, it would be more cost-efficient to give up on the worse performing and slower training networks and focus computational resources on the best network. This algorithm, however, is specifically developed for ensemble models using shared weights. The implementation and variations of these models are shown in Section 3.2.

Contrary to the proof of convergence by the original Hedge algorithm, HBP can not make such claims. In this adapted algorithm, the experts, consisting of classification and hidden layers, change throughout the hedging process. Additionally, the hedging influences the rate of learning for each expert. Due to these factors, the expert that may have been best at the start of training, and would have therefore come out on top with the hedge algorithm, may be outperformed by experts that are better optimized through the backpropagation portion of HBP, and the corresponding α values should adjust accordingly.

3.2 Models

3.2.1 ResNet34

ResNets [He et al., 2016], in its many variants, have been established convolutional neural networks (CNNs) since their initial appearance. This type of network uses skip connections to provide later layers with additional information from previous layers, thereby circumventing the vanishing gradient problem. The skip connections employed by ResNets are, as the name suggests, residual connections. These connections, mathematically speaking, are additions. This means that the input to a block of convolutional layers is put through the block, after which the original input is added to it in order to maintain information from the input that may otherwise be lost. In order for the input to be structurally similar to the output of the block, the skip connection incorporates an identity filter, which applies only the structural changes of the block, such as downsampling, without extracting further features. This sequence of convolutional blocks with residual connections can be used to an arbitrary depth, since the use of residual connections guarantees the deeper network will not have worse performance than a more shallow version.

In this work, ResNet has been implemented using 34 layers. While deeper ResNet models have slightly higher performance, the relative shallowness of each of ResNet34s blocks allows for better comparison between intermediate outputs. This concept will be further explained in Section 3.2.3. The model has an initial convolutional layer using a kernel size of 7, a stride of 2, and a padding of 3. This layer turns the RGB-channel input image into 64 feature maps, after which batch normalization and a max pooling layer, with kernel size 3 and stride 2, are applied. When this initial step is done, the block structure starts. Each of the 4 blocks consists of a number of convolutional layers, each followed by batch normalization. These convolutional layers use kernel size 3, a stride of 1, and a padding of 1. The exception to this is the first layer of blocks 2, 3 and 4, which use a stride of 2 to provide downsampling at the same moment the amount of feature maps double. There are 6, 8, 12 and 6 convolutional layers in each block, in that order, using 64, 128, 256 and 512 feature maps, respectively. After each of these convolutional layers, batch normalization is applied. To the final layer of block 4 average pooling is applied, after which a fully connected layer is used to provide a classification output. It is important to note that the rectified linear unit (ReLU) activation is used throughout the model and that the convolutional layers do not use a bias.

The residual connections occur every 2 convolutional layers during the block structure. These residual connections take the input of the first of the two layers and add this to the output of the second of the two layers. In the case of the first set of layers of blocks 2, 3, and 4, which downsample the input, the input is first put through a convolutional layer with kernel size 1 and stride 2 before being added to the output of the second layer, to accommodate equal dimensions.

The initial convolutional layer, the number of convolutional layers in the block structure and the final classification layer add up to 34 total layers, excluding pooling and normalization layers, giving ResNet34 its name.

3.2.2 DenseNet121

Similar to ResNet variants, DenseNet [Huang et al., 2017b] is also implemented using skip connections to solve the vanishing gradient problem. However, in DenseNet, the skip connections perform a concatenation. At each point between dense blocks, the output feature maps from that block are concatenated with feature maps produced by previous blocks. These very large amounts of feature maps are then used as input for the next dense block. By using this process, the information about the input data is not explicitly preserved. Rather, the model uses lower level features throughout the stages of feature detection, adding increasingly higher level features. As such, the final classification is made using extracted features from multiple layers of complexity.

In this work, the smallest version of DenseNet is used, namely DenseNet121. Similar to the argumentation for ResNet34, this allows for more shallow dense blocks, making the distance between intermediate outputs smaller. The function of this will be explained further in Section 3.2.3. The model has an initial convolutional layer using a kernel size of 7, a stride of 2, and a padding of 3. This layer turns the RGB-channel input image into 64 feature maps, after which batch normalization and a max pooling layer, with kernel size 3 and stride 2, are applied. There is a 4-block structure, similar to ResNet, however these blocks, named dense blocks, function significantly differently. A dense block consists of a specific number of dense layers, with DenseNet121 using 6, 12, 24, and 16 dense layers. Each dense layer consists of a batch normalization layer, a 1×1 convolution, a second batch normalization layer, and a 3×3 convolution with padding 1.

The number of feature maps are determined by the concatenating procedure of DenseNet. This means that the first dense layer of dense block 1 will receive the base 64 feature maps. Then, each dense layer will add to this an amount of features equal to the expansion rate, which is 32. These additional features are concatenated with the input features and used as input for the next dense layer. After the final dense layer of the first dense block, there will be $64 + 6 \times 32 = 256$ feature maps. After a dense block, a transition layer is used to prepare these concatenated features for the next denseblock. The transition layer consists of batch normalization, followed by a 1×1 convolutional layer with an output amount of feature maps equal to half the input features. Finally, an adaptive average pool layer with a kernel size of 2 and stride of 2 provides the downsampling needed for the next dense block. The structure of these dense blocks, in addition to the transition layers provide 128 feature maps as input to block 2, 256 feature maps as input to block 3, and 512 feature maps as input to block 4.

Dense block 4 will output 1024 feature maps. After these are produced, a final batch normalization is applied before a fully connected layer provides the classification output. As with ResNet, all layers use ReLU activation and no bias.

DenseNet121 derives its name from the total number of convolutional layers from the initial convolution (1), dense blocks (116), and transitional layers (3), combined with the fully connected layer (1), for a total of 121 layers.

3.2.3 Multiple Heads Variants

Using ResNet34 and DenseNet121 as a base, the first novel design is a deep CNN using multiple heads. These models are named MHResNet34 and MH-DenseNet121. The structure of these models is shown in Figures 3.1 and 3.2 for flow of data in training and inference, respectively.

These figures show the existence of 3 classifiers. In the literature these would be defined as a classifier head, together with 2 auxiliary heads. However, since the models in this work use all classifier heads equally, the term auxiliary is not used, as it implies lesser importance. In contrast to models such as Inception [Szegedy et al., 2015, Szegedy et al., 2016], where the auxiliary heads are used for training and subsequently disregarded at inference, these models keep all classifiers active at training, validation, and inference.

Having three active classifiers at inference time effectively makes the model an ensemble with no added convolutional layers. The classifiers are implemented as a single fully connected layer, using the features of blocks 2, 3 or 4 as input and using the number of classes as output size. As shown in Table 3.1, the addition of these layers adds negligible amounts of additional model parameters. More specifically, the multiple heads variants have 0.3% and 0.02% additional parameters compared to ResNet34 and DenseNet121, respectively.

To produce the MH variant from existing models such as ResNet34 and DenseNet 121, two fully connected layers are attached at the end of blocks 2 and 3 in addition to the fully connected layer attached to block 4. In order for each classifier to function similarly to the final classifier, adaptive average pooling is applied before attaching the fully connected layers at the same step in each block, which means after the final batch normalization for that block in ResNet34 and before the transition layer for that block in DenseNet121.

As a prelude to the HBP algorithm, the MH variant uses the value of α in both the forward and backward pass. However, α is not yet adapted using the hedge update step. Instead, α is set at an equal value for each classifier. This means each classifier contributes equally to the final classification.

During the backward pass, α is used to update the weights of the model. This is done for each classifier individually, where each classifier's loss only impacts the layers leading up to that classifier. As such, the loss of classifier 1 is backpropagated through the initial convolutional layer, blocks 1 and 2, and classifier 1, updating their weights. The loss of classifier 2 affects the initial convolutional layer, blocks 1 through 3, and classifier 2. Finally, the loss of classifier 3 is used to update the weights of the initial convolutional layer, blocks 1 through 4, and classifier 3, which is the same path of backpropagation as in the original network.

The additional classifiers are nothing more than fully connected layers connecting output features to class predictions. As such, no large amount of additional

Table 3.1: Number of trainable model parameters of all model variants on the CIFAR10 dataset (10 output classes)

Model	Parameters
ResNet34	21289802
MHResNet34	21294962
HBPResNet34	21294965
ThawResNet34	21294965
DenseNet121	6964106
MHDenseNet121	6986290
HBPDenseNet121	6986293
ThawDenseNet121	6986293

parameters is needed, as shown in table 3.1.

3.2.4 HBP Variants

Using the MH model variants with full HBP, with both backpropagation and α updates, produces the HBP model variants. These model variants are identical to the MH variants at initialization. During training, however, the α values are made adaptable, resulting in a model that can learn to use the best parts of the ensemble most effectively. In other words, using the description provided in Section 3.1.1, this allows the best performing classifier to have the biggest impact on shared model weights. Contrary to using HBP in an ensemble without weight sharing, the discrepancy between classifiers will not become an issue in the same manner, as each classifier benefits from the others' learning.

The effect of HBP to train all model weights simultaneously only works in one direction. If classifier 3 performs best, all blocks will learn from this, regardless of the performance of classifiers 1 and 2. However, if classifier 1 or 2 perform best, and α for classifier 3 drops very low, convolutional block 4 will be excluded from learning. To negate this effect a hyper-parameter s was introduced. As explained in section 3.1.1, this values determines both lower and upper bound of the network. Due to a lower bound for α, no classifier will ever be fully disconnected from learning, along with the attached convolutional blocks.

3.2.5 Thawing HBP Variants

During development, a possible downside of adapting α throughout the entire training process was exposed. If at any point during training a single classifier would outperform the others, for that classifier α would go up dramatically, while α would diminish to the lower bound for all other classifiers. If this happened too early in training, the resulting behaviour would be a complete focus of the model on a single classifier. This problem was somewhat solved with the introduction of the s parameter. However, a different solution was also implemented. In the Thawing model variants, the lower bound s is not defined by a single value, but

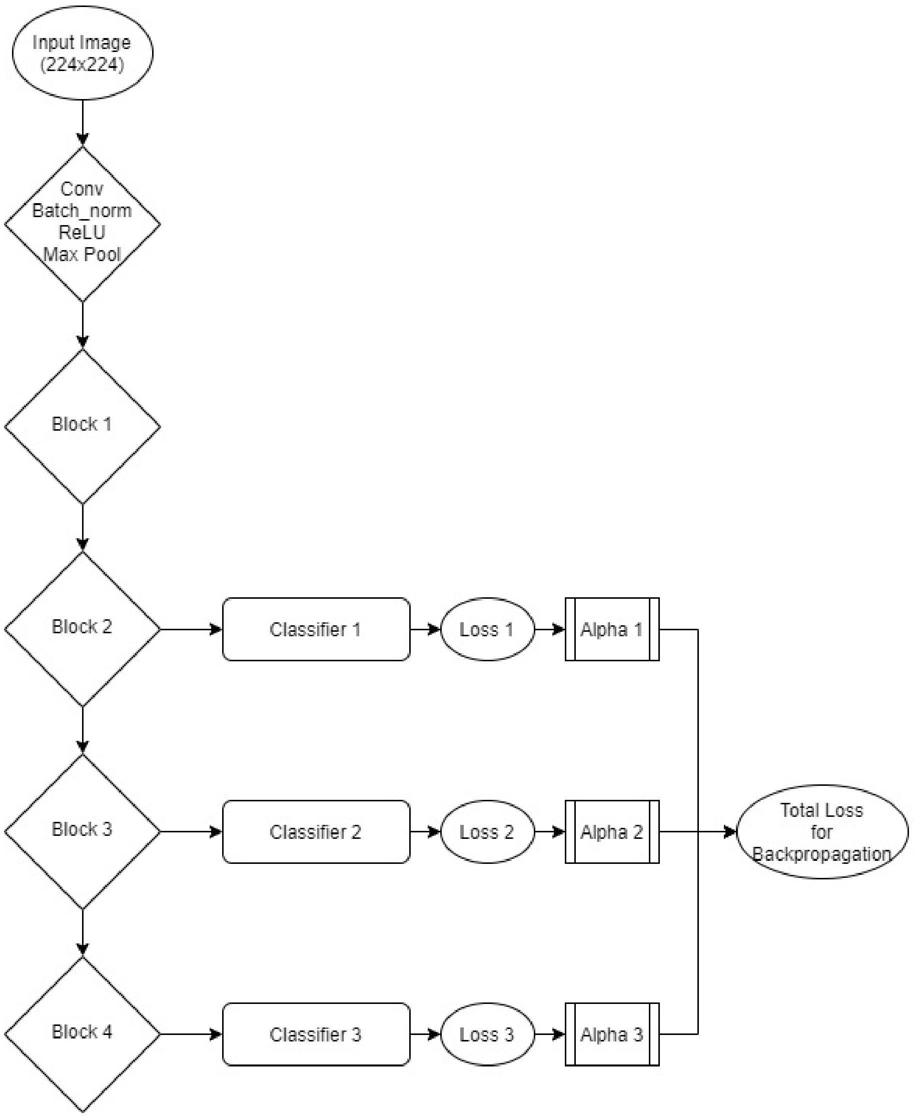

Figure 3.1: Abstract model structure for all models with multiple classifiers (MH-variants, HBP-variants, Thaw-variants). Data flow represents loss by individual classifiers, which is used to update separate parts of the network.

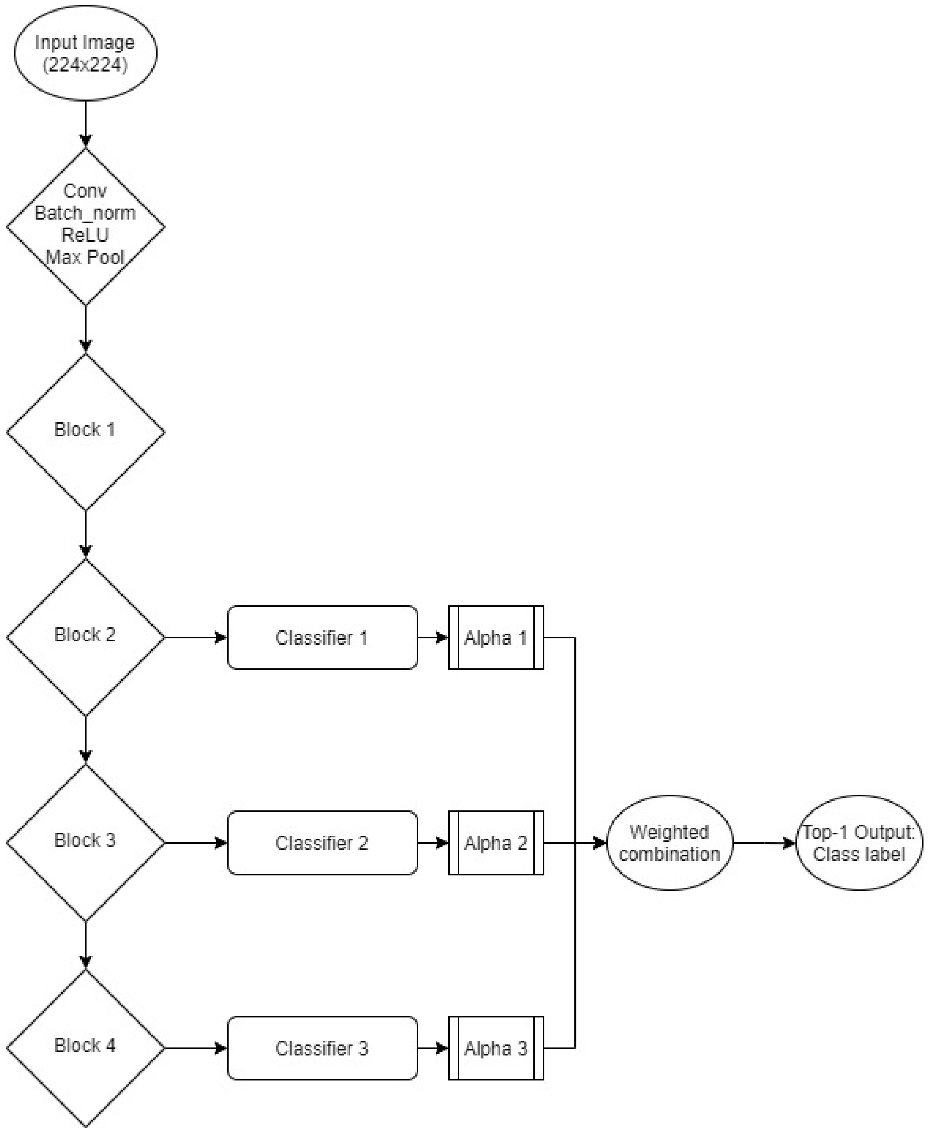

Figure 3.2: Abstract model structure for all models with multiple classifiers (MH-variants, HBP-variants, Thaw-variants). Data flow represents shows combined final output, used to determine accuracy of the model.

rather scaled over time according to Equation 3.4

$$s = \theta \times \gamma^t \tag{3.4}$$

Equation 3.4 introduces 2 additional hyper-parameters, namely base value θ and discount factor γ. Additionally, t represents the epoch. Assuming $\theta > 1$ and $0 < \gamma < 1$ produces a thawing effect. At $t = 0$, s will be larger than 1, which negates any attempts to update the models α, as described in Section 3.1.1. The discount factor will lower s over time, allowing α to be updated after a set amount of time. In effect, the model gradually shifts from a MH variant to a HBP variant. To make sure the same issues do not arise as described for HBP variants, and to ensure the behaviour resembles HBP variants after thawing, the s is itself lower bound to a predetermined value identical to HBP variants.

3.3 Data

To test the effectiveness of each of these model variants, two classification datasets were used, each with their own purpose. CIFAR10 [Krizhevsky, 2009] provides an opportunity to train from scratch with a reasonable timeframe, since the dataset is simplistic, with a limited number of output classes and large inter-class differences. In contrast the Papilionidae dataset [Naturalis Biodiversity Center, 2021] provided by Naturalis is significantly more complex, with a larger amount of classes and very subtle inter-class differences. This allows for the use of transfer learning for fine-grained classification, effectively testing the models on two comparable, but distinct, tasks.

3.3.1 CIFAR10

The CIFAR10 dataset contains 60000 images, spread over 10 classes. The class spread is even, with 6000 images per class. The dataset is initially split up into 50000 train images and 10000 test images. The 10 classes are made up for 4 distinct vehicle types (airplane, car, ship, truck) and 6 distinct animal types (bird, cat, deer, dog, frog, horse). The dataset creators claim no overlap between classes, specifically for classes that could be ambiguous such as car and truck. The images are 32 by 32 pixels and in color.

3.3.2 Naturalis Papilionidae

The Papilionidae dataset provided by Naturalis contains 8243 images of butterflies. These images are labeled with many different attributes such as origin, date of collection, and who identified them. The most important labels, however, are genus, species and subspecies. Using these labels as class labels, the dataset can be used for increasingly difficult fine-grained classification, with 11, 79 and 112 classes, respectively. For this work, the subspecies label, with 112 classes, is used. The color images are of differing size and range from 400 to 800 pixels in either dimension. The dataset contains a csv file for all relevant labeling data.

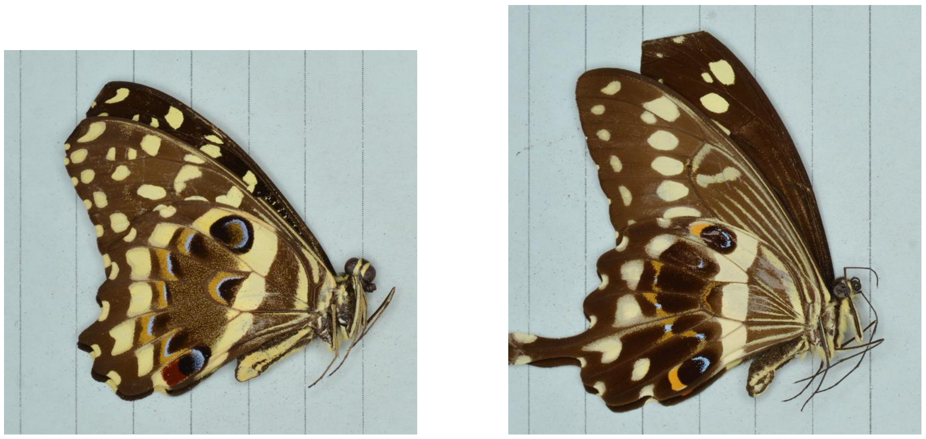

Figure 3.3: Two examples from the Papilionidae dataset. The left image is of the class Papilionidae-Papilio-demodocus-demodocus. The image on the right is of the class Papilionidae-Papilio-lormieri

However, there are more entries than images provided. As such, the csv file needs to be filtered to exclude all entries without a corresponding image.

Chapter 4

Experimental Setup

Using the models and data described in Chapter 3, the research questions from Section 1.1 will be answered using several experiments. To allow for reproduction of results, all steps of preprocessing, data augmentation, and hyper parameter optimization will be described in this chapter.

4.1 Preprocessing

In order to prepare the data for optimal use in a convolutional neural network several simple steps are taken. First, the data is resized to 224 by 224 pixels, which is considered optimal input size [He et al., 2016, Huang et al., 2017b]f for ResNets and DenseNets. Second, the image RGB channels are rescaled to a range of 0 to 1, instead of 0 to 255. Third, the images are normalized, meaning that, for each color channel, the mean is subtracted before dividing by the standard deviation. Third, the labels are produced. For CIFAR10, a dataset readily available in the PyTorch framework [Paszke et al., 2019], the labels are ready made integer values from 0 to 9. For the Papilionidae dataset, the labels are taken from the *class_infra_species* entry of the csv file. These are then encoded from string values to integers ranging from 0 to 111.

4.1.1 Data Augmentation

The application of data augmentation to produce a more generalizable model is standard practice. The augmentation is readily available in the PyTorch framework [Paszke et al., 2019], allowing for quick implementation. The augmentations used are: horizontal flip with a random chance of 0.5, rotation with a range of -20 to 20 degrees, a color jitter transformation with ranges for brightness, contrast, saturation, and hue all set to [0.8, 1.2] of current values, and an affine transformation with shear in the x-axis with a range of -10 to 10 degrees in the RGB channels.

These augmentations are applied continuously. Whenever a batch is selected for training, all augmentations are applied with their corresponding ranges. This means that it is statistically highly unlikely that the same variation of the same image is seen twice.

4.2 Hyper Parameter Optimization

The models require hyperparameter optimization on multiple aspects. Some values have been copied from the original implementation of the networks, such as loss function, activation function, and optimizer. Others required manual optimization due to the distinct changes in model structure. Early in development, there was a strong indication that the variant networks worked better with a learning rate that was significantly higher than that of networks with a single classifier. Due to this, the learning rates are set to 0.01 for DenseNet121 and ResNet34, and 0.1 for all variant networks. It makes intuitive sense to use these higher learning rates, as the use of α in HBP means all losses used to calculate weight updates are multiplied by a factor smaller than 1.

Of further interest are the hyperparameters introduces specifically by the HBP algorithm, both in its pure form and the thawing variant. First, the value s is introduced as a lower bound for α. Tables 4.1 and 4.2 show the results for different s-values for HBPResNet34 and DenseNet121, respectively. Here, we see 2 different patterns. For HBPResNet34 there is a small upward trend in mean accuracy with an increasing value s, with the best performance found at $s = 1.0$. In contrast, for DenseNet34, a value of $s = 0.50$ performs best. What this shows is an indication that for ResNet34 variants, the MH model simply performs better than the HBP model, since $s = 1.0$ for HBPResNet34 makes the model identical in behaviour to MHResNet34, due to the outcome of Equation 3.3 for the upper bound and $\frac{s}{N}$, both being 0.33. With the value of 0.5 for s being a suitable choice for HBPDenseNet121 and the desire to experiment with HBPResNet34 as a distinct model from MHResNet34 throughout other experiments, the s value for all HBP and Thaw models has been set to 0.5.

Table 4.1: Hyper parameter optimization on s: Results for HBPResNet34 on the CIFAR10 dataset, using 10% (4500) training images.

s-value	Fold 1	Fold 2	Fold 3	Fold 4	Mean
0.00	72.3	72.0	73.9	73.5	72.9
0.25	74.2	73.6	73.6	74.0	73.9
0.50	75.9	75.5	75.0	72.5	74.7
0.75	**76.4**	75.0	76.0	76.4	76.0
1.00	76.0	**75.7**	**76.5**	**76.7**	**76.2**

Second, the combination of θ and γ used in thawing variants is explored. Table 4.3 shows what values used for θ and γ lead to which thawing behavior. The results for each of these combinations are shown in Table 4.4 and 4.5, for ThawResnet34 and ThawDenseNet121, respectively. For ThawResNet34, the

Table 4.2: Hyper parameter optimization on s: Results for HBPDenseNet121 on the CIFAR10 dataset, using 10% (4500) training images.

s-value	Fold 1	Fold 2	Fold 3	Fold 4	Mean
0.00	73.9	72.0	73.4	73.9	73.3
0.25	75.5	74.4	75.2	75.4	75.1
0.50	**76.5**	74.4	74.9	**76.4**	**75.5**
0.75	75.7	**74.7**	**75.8**	75.7	**75.5**
1.0	75.0	74.3	74.8	75.3	74.9

parameters leading to the shortest frozen periods (($\theta = 1.5$, $\gamma = 0.95$), ($\theta = 3.0$, $\gamma = 0.95$)) have sub-optimal results. The other behaviours, with longer periods of frozen α, have marginally better performance. This leads to a similar conclusion as for the s parameter, where a longer frozen time has better performance, because it more closely resembles MHResNet34. For ThawDenseNet121, the shorter freezing times marginally outperform longer freezing times, with top performance for ($\theta = 3.0$, $\gamma = 0.95$). Similar to the argumentation used for s, these values are used for both ThawResNet34 and ThawDenseNet121, to allow a clear distinction between ThawResNet34 and MHResNet34 in further experiments.

Table 4.3: Thawing behaviour in epochs depending on θ and γ values. Column < 1.0 notes when α becomes changeable as it drops below the 1.0 threshold. Column < 0.5 notes when the thawing process is complete, as s is still lower bound to 0.5 as with regular HBP models.

θ	γ	< 1.0	< 0.5
1.5	0.98	21	55
1.5	0.95	8	22
3.0	0.95	22	35
10.0	0.95	45	59
150	0.90	45	55

Table 4.4: Hyper parameter optimization on θ/γ: Results for ThawResNet34 on the CIFAR10 dataset, using 10% (4500) training images. Acc shows final validation accuracy, t shows epoch of convergence.

Parameters		Fold 1		Fold 2		Fold 3		Fold 4		Mean	
θ	γ	Acc	t	Acc	t	Acc	t	Acc	t	Acc	t
1.5	0.98	75.1	88	75.1	86	75.6	123	74.0	80	75.0	94.25
1.5	0.95	76.2	127	74.0	75	74.0	**83**	74.6	83	74.7	92
3.0	0.95	73.5	**69**	73.8	135	**76.4**	97	74.8	102	74.6	100.75
10.0	0.95	76.0	94	75.0	**73**	76.0	113	74.7	**72**	75.4	**88**
150.0	0.90	**76.9**	108	**75.2**	75	74.9	88	**76.6**	116	**75.9**	96.75

Table 4.5: Hyper parameter optimization on θ/γ: Results for ThawDenseNet121 on the CIFAR10 dataset, using 10% (4500) training images. Acc shows final validation accuracy, t shows epoch of convergence.

Parameters		Fold 1		Fold 2		Fold 3		Fold 4		Mean	
θ	γ	Acc	t	Acc	t	Acc	t	Acc	t	Acc	t
1.5	0.98	**75.9**	119	73.8	**83**	75.6	80	73.9	**69**	74.8	87.75
1.5	0.95	75.6	100	75.2	102	**76.0**	92	75.3	91	75.5	96.25
3.0	0.95	75.2	95	75.5	121	75.8	87	**77.7**	105	**76.1**	102
10.0	0.95	75.5	**71**	**77.1**	150	73.3	127	75.8	106	75.4	113.5
150.0	0.90	**75.9**	77	73.5	106	75.3	**82**	75.9	83	75.2	**87**

For all these hyper parameter sweeps, a shortened version of k-folds cross-validation is applied. For each fold, a different validation set is selected, consisting of 10% of the training data. After selecting a validation set, 10% is taken from the remaining data and used for training. This is then repeated for 4 folds. This reduced cross-validation is used to give a broader sense of optimal parameters without significantly high computing costs in the hyper parameter optimization stage.

4.3 Experimental settings

The effectiveness of the models will be evaluated using three distinct experimental settings. These are:

- Trained from scratch on the CIFAR10 dataset.
- Trained from scratch on the Papilionidae dataset.
- Fine-tuned on the Papilionidae dataset, using pretrained weights from ImageNet.

For all these settings, the same procedure is used. K-folds cross-validation is used with $k = 5$. First, the data is shuffled after which 10% of the data is taken as validation. Of the remaining 90% of data, a percentage of training images is used. For each setting and each fold, the amount of data used to train is 5%, 10%, 20%, 50% and 100%, to allow for comparison of methods with differing amounts of data.

In total, each of the three experimental setting requires 5 different segments of data, with 5 folds per data split, for a total of 75 training/validation experiments per model. Applying these to all 8 models produces a total of 600 training/validation experiments.

4.3.1 Early Stopping

In order to counteract stagnation in training the model, two measures are implemented. First is a learning rate schedule. This schedule divides the learning rate with a factor 10 if validation accuracy has not improved for 5 epochs. This allows the model to search broadly at first, with a high learning rate, and eventually search more gradually to reach an optimum. In addition, to avoid overtraining, early stopping is used to cut off training when the validation accuracy has not improved for 20 epochs.

4.4 Specifications

4.4.1 Hardware

The hardware used to run the experiments and produce results is provided by the HPC center of the university of Groningen. The cluster, named Peregrine, provides high performance CPU and GPU nodes, which are very effective for large parallelizable operations, such as neural network optimization. The GPU nodes consist of a virtualized Intel Xeon Gold 6150 CPU and a NVIDIA V100 GPU. Using this hardware, the experiments ran for approximately 5 days per model per experimental setting, for all folds and data splits.

4.4.2 Software

The models and all data processing is implemented using Python 3.6 and PyTorch 1.6 [Paszke et al., 2019]. Code is stored on github and available on request.

Chapter 5

Results

In this chapter the results of the proposed experiments are presented. These results are analyzed in a variety of ways, in order to provide the insights needed to answer the research questions. To reiterate, the research questions of this work are:

1. Does a model with multiple heads perform better than the base model?
2. Do the additional heads increase performance when active during inference, in contrast to only using additional heads during training?
3. Does the hedge backpropagation (HBP) algorithm successfully optimize weights between output layers of a model with multiple heads?
4. Does freezing the weights between output layers of the model at the start of training influence the performance of the model? Does this significantly differ from the MH or HBP models?
5. Do the behaviours of these models change when using differing amounts of data?

The raw data of all experimental settings is shown in Tables 7.1 to 7.15. These tables show the validation accuracy at time of convergence with early stopping. Since the folds for each experiment are non-randomly split for all models and settings, the individual fold results, as well as the mean of the 5 folds, can be compared for further analysis.

Table 5.1: Results on CIFAR10 for all models, with different numbers of training samples. Shown is the mean and standard deviation for each test scenario. The best performing model variant per number of samples is shown in bold.

Model	5%		10%		20%		50%		100%	
ResNet34	62.1	1.013	72.2	0.888	79.1	0.542	87.2	0.242	91.3	0.233
MHResNet34	**66.5**	1.323	**75.9**	0.492	82.1	0.816	**89.2**	0.531	92.3	0.264
HBPResNet34	64.8	1.825	75.3	1.070	**82.6**	0.508	88.8	0.603	**92.7**	0.331
ThawResNet34	66.3	2.307	**75.9**	1.126	82.1	0.294	89.0	0.777	**92.7**	0.219
Densenet121	63.7	1.132	72.3	1.689	79.8	0.723	87.6	0.534	91.5	0.185
MHDenseNet121	**66.1**	1.338	75.2	1.216	**83.0**	0.306	89.1	0.456	92.8	0.141
HBPDenseNet121	64.6	1.588	74.8	1.538	82.8	0.540	88.8	0.791	**93.0**	0.141
ThawDenseNet121	64.1	1.579	**75.6**	0.801	82.8	0.496	**89.2**	0.500	92.8	0.417

Table 5.2: Results on Papilionidae for all models without transfer learning, with different numbers of training samples. Shown is the mean and standard deviation for each test scenario. The best performing model variant per number of samples is shown in bold.

Model	5%		10%		20%		50%		100%	
ResNet34	**81.2**	2.039	**85.1**	1.487	88.6	1.529	90.9	1.346	92.5	0.508
MHResNet34	76.8	5.200	82.3	4.960	88.4	1.245	**91.0**	1.385	92.5	0.891
HBPResNet34	80.1	1.713	84.6	1.240	88.0	1.422	90.8	1.391	92.3	0.774
ThawResNet34	81.0	2.715	84.4	2.066	**89.0**	0.865	90.6	1.357	**92.6**	0.723
Densenet121	81.8	1.608	84.9	1.662	**89.0**	1.260	91.0	0.980	**93.2**	0.540
MHDenseNet121	81.5	1.066	84.3	2.397	**89.0**	1.188	90.8	1.261	92.6	1.019
HBPDenseNet121	82.1	0.989	**85.7**	1.488	88.8	1.340	**91.6**	0.974	92.8	0.776
ThawDenseNet121	**82.7**	1.109	84.5	1.392	87.6	1.198	90.7	1.366	93.0	0.668

Table 5.3: Results on Papilionidae for all models with transfer learning, with different numbers of training samples. Shown is the mean and standard deviation for each test scenario. The best performing model variant per number of samples is shown in bold.

Model	5%		10%		20%		50%		100%	
ResNet34	**86.3**	1.108	**88.7**	1.192	**90.4**	0.910	**92.5**	0.849	**94.3**	0.584
MHResNet34	82.3	2.098	83.6	2.199	87.8	2.294	91.3	1.078	93.5	0.686
HBPResNet34	80.7	1.578	82.8	2.701	87.6	1.216	90.5	1.371	92.7	1.040
ThawResNet34	83.8	1.579	83.4	2.345	88.8	1.345	91.6	0.988	93.7	0.711
Densenet121	**86.9**	1.083	**89.0**	1.111	**90.3**	1.240	**93.0**	0.674	94.5	0.540
MHDenseNet121	86.2	1.243	87.8	1.347	**90.3**	1.391	92.8	0.582	**94.7**	0.462
HBPDenseNet121	86.0	0.999	87.5	1.212	**90.3**	1.246	92.2	1.060	94.1	0.449
ThawDenseNet121	85.6	0.806	88.3	1.508	**90.3**	1.185	92.7	0.812	94.6	0.341

5.1 Does a model with multiple heads perform better than the base model?

The comparison of base models, i.e. ResNet34 and DenseNet121, and the MH variants, i.e. MHResNet34 and MHDenseNet121, is done by performing a pairwise t-test using the full experimental results as shown in Tables 7.1 to 7.15. The results of these t-tests are shown in Tables 5.4 and 5.5. These tests are performed on the full range of experiments and on each experimental setting individually.

The t-tests show that both MHResNet34 and MHDenseNet121 outperform their respective base variants with an increase of roughly 2.3% in accuracy. Tables 5.4 and 5.5 also show that these increase are not universal. For the experiments performed on the Papilionidae dataset, both with and without transfer learning, the difference in performance is either insignificant, or in favor of the base variant. These statistical findings can be verified from Tables 5.1, 5.2, and 5.3, where MH variants score higher than base variants in 10 out of 10 scenarios with CIFAR10, whereas the MH variants perform worse or equal in other settings.

This shows that the addition of multiple heads to the network base variants can increase performance, but only in specific circumstances.

Table 5.4: Comparison of mean performance across all data splits and folds between ResNet34 and MHResNet34. A pairwise t-test is used to determine a significant difference in means, where difference is defined as mean of MHResnet34 - mean of ResNet34. A positive difference means MHResNet34 outperforms Resnet34, and vice versa. The resulting p-value relates to the null hypothesis that ResNet34 and MHResNet34 are not significantly different.

Experiment	p-value	Diff. in means
All	0.278	-0.448
CIFAR10	<0.0001	2.316
Papilionidae, untrained	0.074	-1.436
Papilionidae, pretrained	<0.0001	-2.744

5.1.1 Why does the difference in performance on CIFAR10 not translate to Papilionidae, both with and without transfer learning?

From Tables 5.4 and 5.5, it is clear that there is a noticeable increase in performance between MH variants and base networks specifically for experiments on CIFAR10. However, this does not translate to experiments on the Papilionidae dataset, both with and without transfer learning. To identify possible causes for this discrepency, Figures 5.1 through 5.6 show a comparison of MH variants and base networks. Accuracy, training loss, and validation loss are plotted over

Table 5.5: Comparison of mean performance across all data splits and folds between DenseNet121 and MHDenseNet121. A pairwise t-test is used to determine a significant difference in means, where difference is defined as mean of MHDenseNet121 - mean of DenseNet121. A positive difference means MHDenseNet121 outperforms DenseNet121, and vice versa. The resulting p-value relates to the null hypothesis that DenseNet121 and MHDenseNet121 are not significantly different.

Experiment	p-value	Diff. in means
All	0.0155	0.00524
CIFAR10	<0.0001	2.32
Papilionidae, untrained	0.301	-0.332
Papilionidae, pretrained	0.0048	-0.412

the entire training sequence. These figures show that on CIFAR10, Figures 5.1 and 5.2, the training loss for both models approaches 0 as the number of epochs increases. However, it does so more gradually for the MH variants. This allows the MH variants to update weights more gradually, whereas the base models are cut off from training sooner due to early stopping to prevent overfitting. In other words, a claim could be made that the additional classifiers provide a regularizing effect that allows the model to generalize better, resulting in better performance.

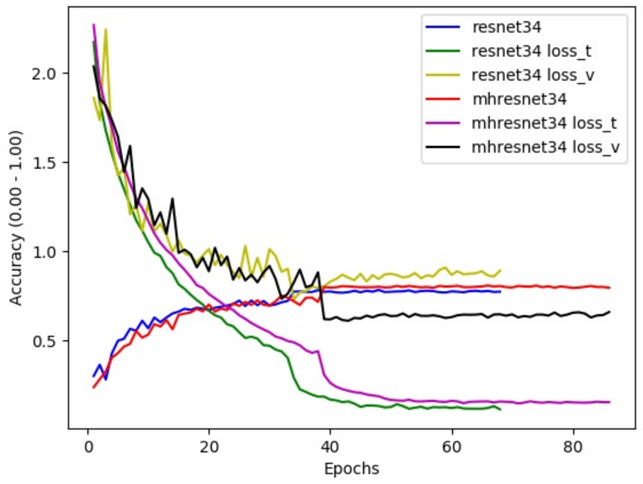

Figure 5.1: Validation accuracy, training loss, and validation loss of ResNet34 and MHResNet34 on CIFAR10.

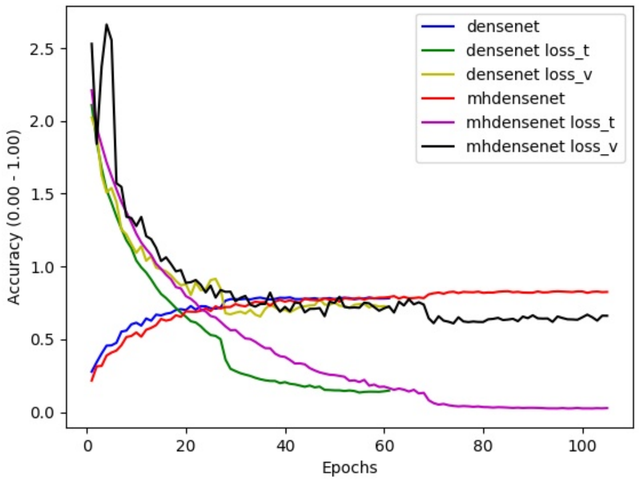

Figure 5.2: Validation accuracy, training loss, and validation loss of DenseNet121 and MHDenseNet121 on CIFAR10.

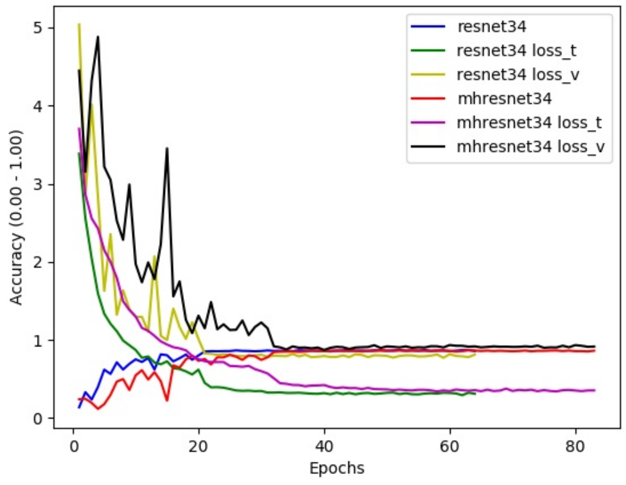

Figure 5.3: Validation accuracy, training loss, and validation loss of ResNet34 and MHResNet34 on Papilionidae, untrained.

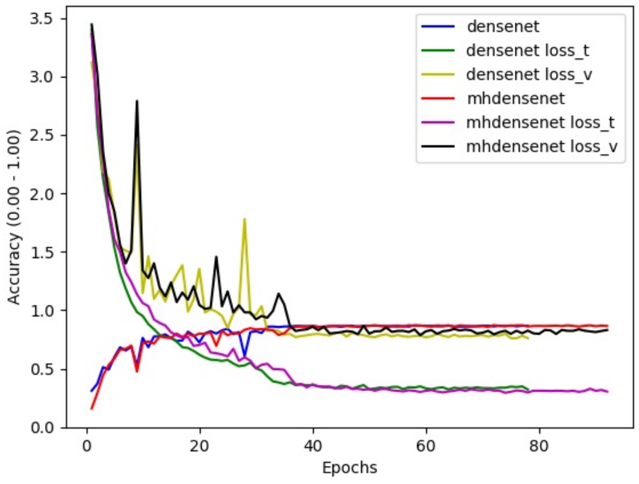

Figure 5.4: Validation accuracy, training loss, and validation loss of DenseNet121 and MHDenseNet121 on Papilionidae, untrained.

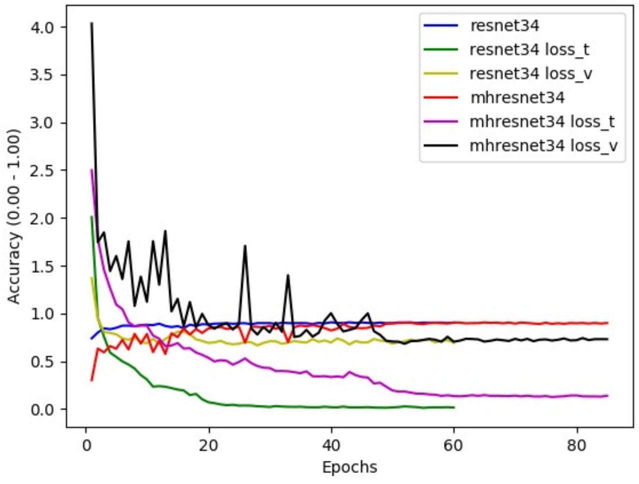

Figure 5.5: Validation accuracy, training loss, and validation loss of ResNet34 and MHResNet34 on Papilionidae, pretrained.

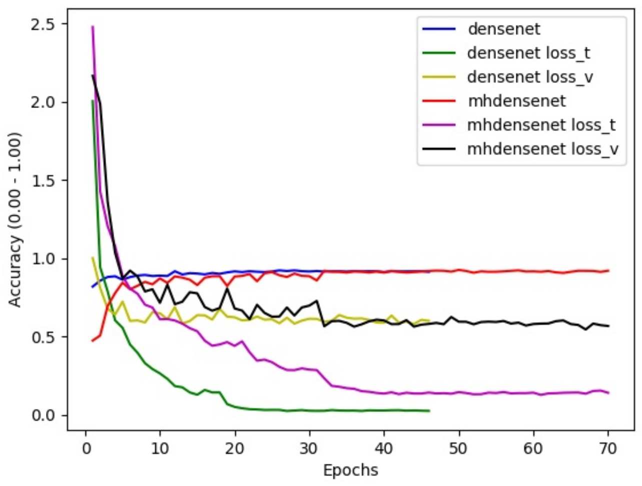

Figure 5.6: Validation accuracy, training loss, and validation loss of DenseNet121 and MHDenseNet121 on Papilionidae, pretrained.

5.2 Do the additional heads increase performance when active during inference?

Assuming the increase in performance as fact, which shows from the experiments on CIFAR10 in research question 1, the follow-up question arises of whether this increase in performance is due to support from the additional classifiers during training or whether the additional classifiers increase performance when used during inference. To explore this, a comparison can be made between the output of classifier 3, and the combined weighted output of the model as a whole. Figures 5.7 through 5.12 show the curve of the validation accuracy throughout training for a single experimental run for each of the three experimental settings.

For Figures 5.7 and 5.8, the combined output of the model is higher than the output of classifier 3 for large portions of the training process, albeit by a small amount. This effect is more distinct for MHResNet34 than for MHDenseNet121. This does not hold for Papilionidae, both with and without transfer learning, as shown in Figures 5.9 through 5.12.

This minor, yet noticeable, increase in performance of the combination model over classifier 3 means that keeping classifiers 1 and 2 intact during inference and combining the outputs of all classifiers does contribute to a higher accuracy for the total model. In deep supervision, the auxiliary heads (e.g. classifiers 1 and 2), are disabled on inference and so these results show that this decision is not universally optimal.

The pattern does not persist in experiments on Papilionidae. This is explain-

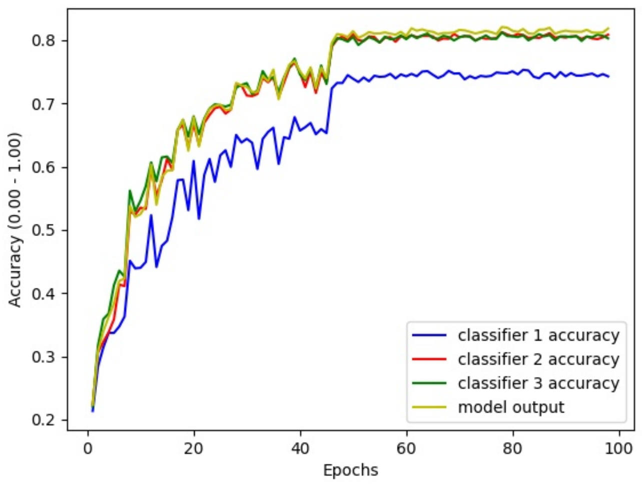

Figure 5.7: Results for MHResNet34 for individual classifiers on 20% of training data for CIFAR10. Shown is the validation accuracy over all training epochs for each individual classifier and the combined weighted output.

able, due to the fact that for these experiments, the base models performed better than any MH variant. As such, it is understandable that a MH variant is optimal when approaching the base model as closely as possible, which means that the final classifier will be trained fully, neglecting classifiers 1 and 2.

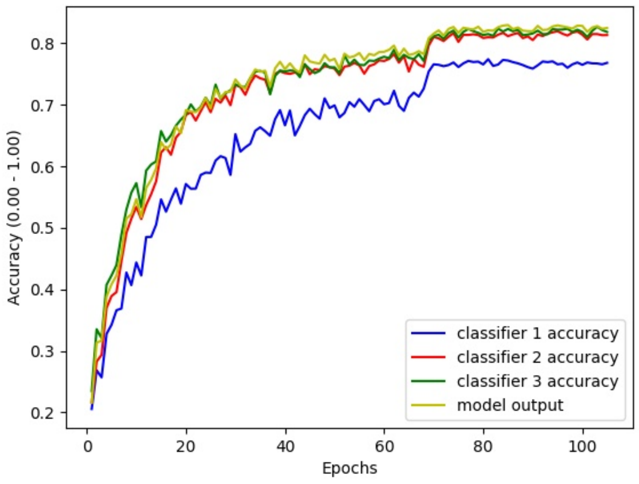

Figure 5.8: Results for MHDenseNet121 for individual classifiers on 20% of training data for CIFAR10. Shown is the validation accuracy over all training epochs for each individual classifier and the combined weighted output.

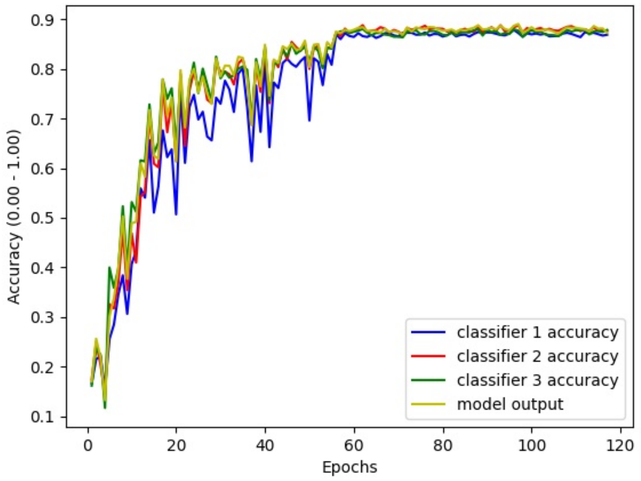

Figure 5.9: Results for MHResNet34 for individual classifiers on 20% of training data for Papilionidae, untrained. Shown is the validation accuracy over all training epochs for each individual classifier and the combined weighted output.

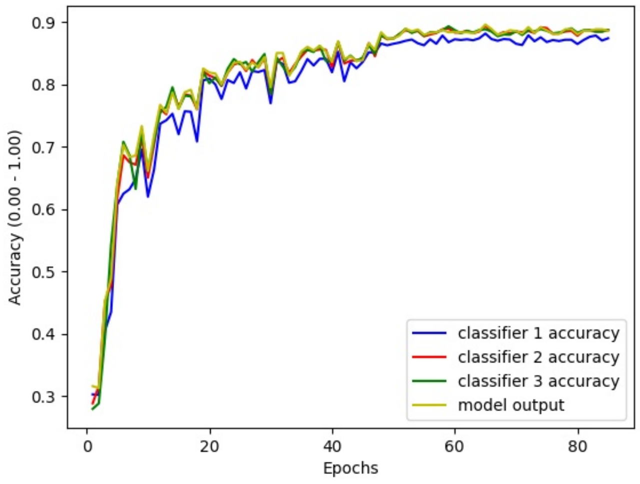

Figure 5.10: Results for MHDenseNet121 for individual classifiers on 20% of training data for Papilionidae, untrained. Shown is the validation accuracy over all training epochs for each individual classifier and the combined weighted output.

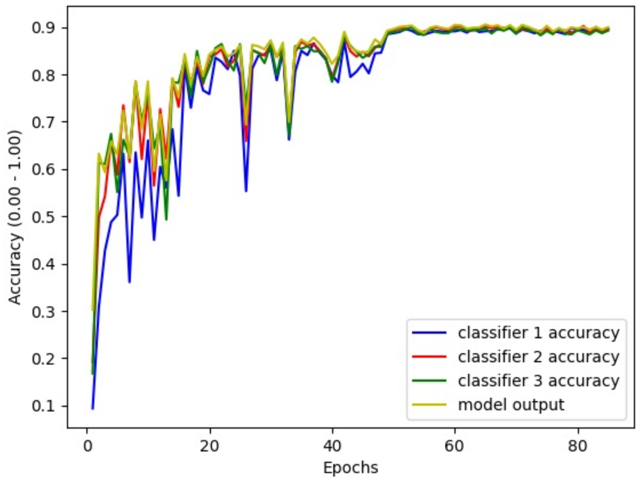

Figure 5.11: Results for MHResNet34 for individual classifiers on 20% of training data for Papilionidae, pretrained. Shown is the validation accuracy over all training epochs for each individual classifier and the combined weighted output.

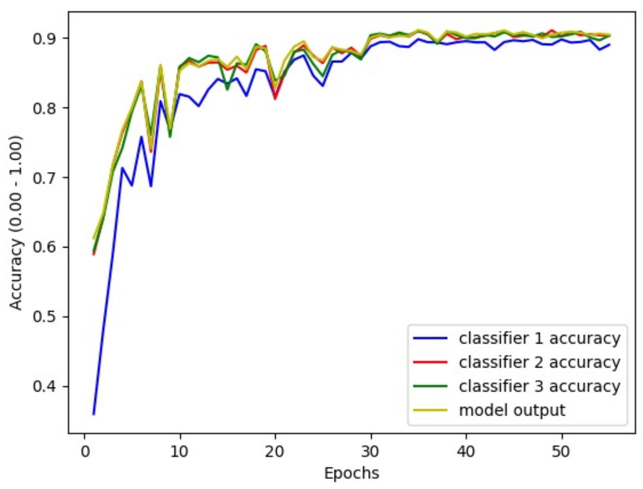

Figure 5.12: Results for MHDenseNet121 for individual classifiers on 20% of training data for Papilionidae, pretrained. Shown is the validation accuracy over all training epochs for each individual classifier and the combined weighted output.

5.3 Does the hedge backpropagation (HBP) algorithm successfully optimize weights between output layers of a model with multiple heads?

To determine the effectiveness of the HBP algorithm in optimizing the use of multiple classifiers, the performance of the HBP variants in relation to the MH variants is analyzed. This is done by comparing performance in each individual experiment as well as the total of all experiments. Comparison is done using a pairwise t-test, with the same argumentation as in Section 5.1.

An example of HBP behavior is shown in Figures 5.13 through 5.18. In these figures, the α and validation accuracies of each individual classifier, as well as the combined model, are plotted over time.

Neither numerical nor graphical results show a significant improvement from the use of HBP, compared to the MH variants. From these results it follows that HBP does not contribute positively to the ensemble structure introduced with the MH variants. Additionally, the Figures show the α values reach the boundary values without fail. In each case, α_3, which corresponds with classifier 3, reaches the maximum, while the others reach the minimum. This can mean one of two things. First, the third classifier is favored in each case, because the encoder structure of the network lends itself most optimally for the third classifier. In other words, the third classifier has the best performance in each case and thus the highest α value. This effect can be further emphasized by the iterative training scheme. In contrast, the hedge algorithm is initially used for online learning, using each case once. Due to the iterative nature of the training scheme, a favored classifier will be pushed to the maximum boundary value by continually performing best. A second explanation for the extremes in α values is the interaction between model weight updates and α updates. A classifier favored at the start of training will train quicker, reaching a better performance earlier. As a consequence of this, the α values will be pushed more in favor of this better performing classifier, allowing it to train more effectively, repeating the process.

Table 5.6: Comparison of mean performance across all data splits and folds between MHResNet34 and HBPResNet34. A pairwise t-test is used to determine a significant difference in means, where difference is defined as mean of HBPResnet34 - mean of MHResNet34. A positive difference means HBPResNet34 outperforms MHResnet34, and vice versa. The resulting p-value relates to the null hypothesis that MHResNet34 and HBPResNet34 are not significantly different.

Experiment	p-value	Diff. in means
All	0.772	-0.085
CIFAR10	0.174	-0.392
Papilionidae, untrained	0.188	0.952
Papilionidae, pretrained	0.049	-0.816

Table 5.7: Comparison of mean performance across all data splits and folds between MHDenseNet121 and HBPDenseNet121. A pairwise t-test is used to determine a significant difference in means, where difference is defined as mean of HBPDenseNet121 - mean of MHDenseNet121. A positive difference means HBPDenseNet121 outperforms MHDenseNet121, and vice versa. The resulting p-value relates to the null hypothesis that MHDenseNet121 and HBPDenseNet121 are not significantly different.

Experiment	p-value	Diff. in means
All	0.596	-0.081
CIFAR10	0.115	-0.488
Papilionidae, untrained	0.071	0.560
Papiolionidae, pretrianed	0.011	-0.316

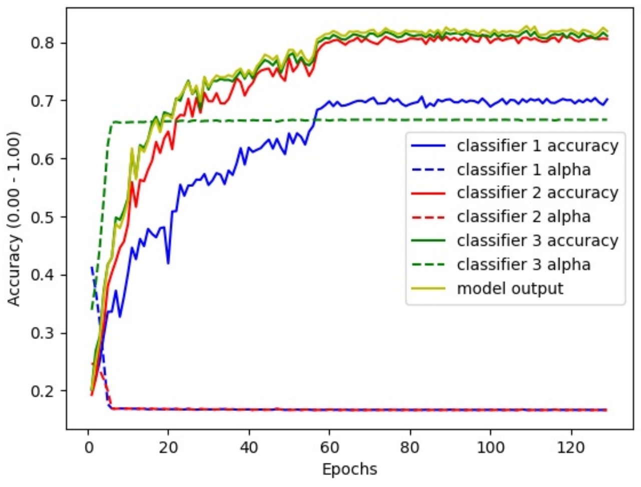

Figure 5.13: Results for HBPResNet34 for individual classifiers on 20% of training data for CIFAR10. Shown is the validation accuracy over all training epochs for each individual classifier and the combined weighted output.

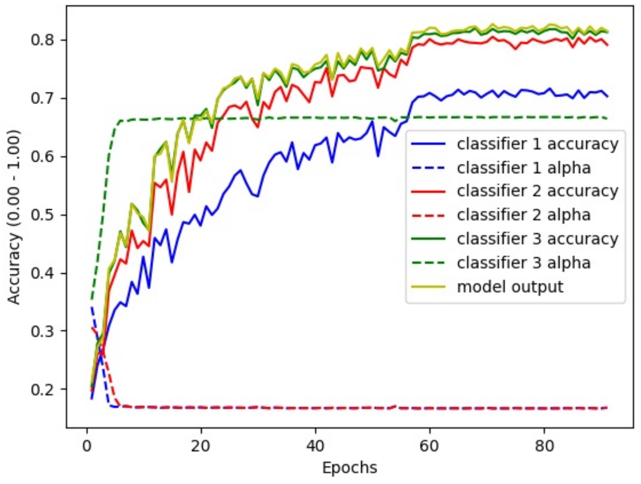

Figure 5.14: Results for HBPDenseNet121 for individual classifiers on 20% of training data for CIFAR10. Shown is the validation accuracy over all training epochs for each individual classifier and the combined weighted output.

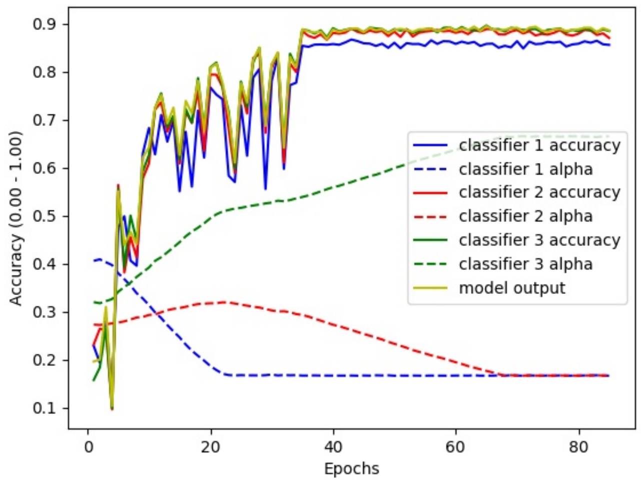

Figure 5.15: Results for HBPResNet34 for individual classifiers on 20% of training data for Papilionidae, untrained. Shown is the validation accuracy over all training epochs for each individual classifier and the combined weighted output.

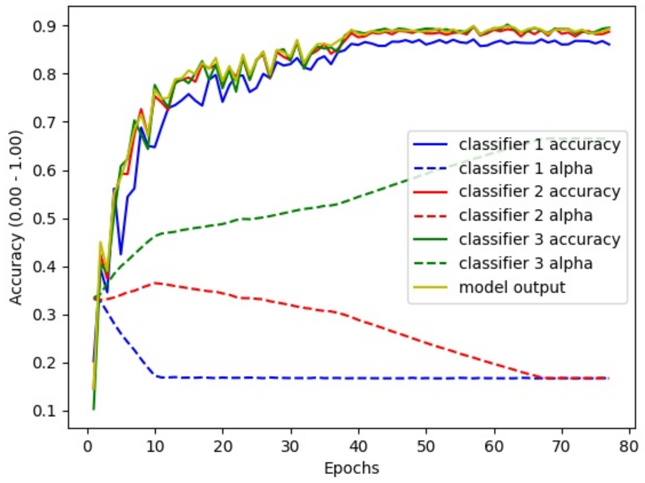

Figure 5.16: Results for HBPDenseNet121 for individual classifiers on 20% of training data for Papilionidae, untrained. Shown is the validation accuracy over all training epochs for each individual classifier and the combined weighted output.

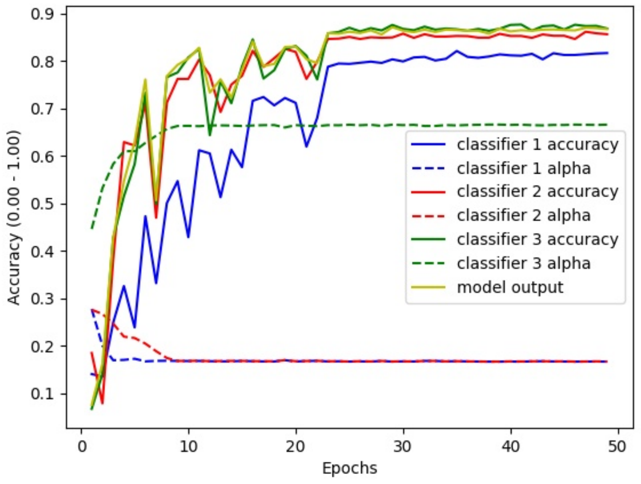

Figure 5.17: Results for HBPResNet34 for individual classifiers on 20% of training data for Papilionidae, pretrained. Shown is the validation accuracy over all training epochs for each individual classifier and the combined weighted output.

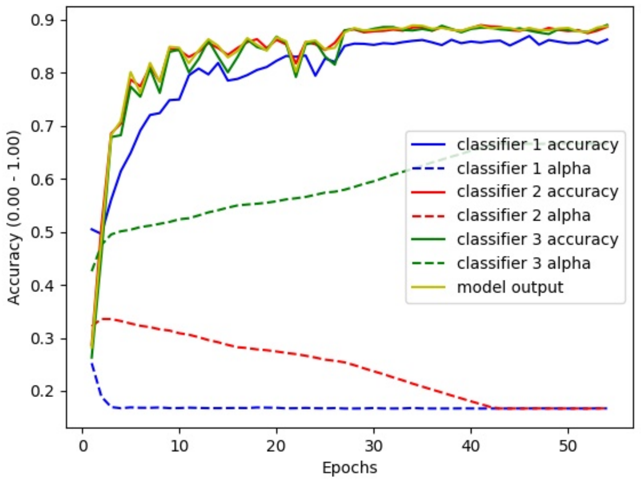

Figure 5.18: Results for HBPDenseNet121 for individual classifiers on 20% of training data for Papilionidae, pretrained. Shown is the validation accuracy over all training epochs for each individual classifier and the combined weighted output.

5.4 Does freezing the weights assigned to classifiers in the model at the start of training influence the performance of the model?

To determine whether postponing the use of HBP to later in the training process by the use of a thawing approach to α, the comparison between the MH variants and the Thaw variants is analyzed using the paired t-test. These comparisons are similar in nature to previous comparisons, using the results of individual experiments, as well as the combination of these experiments. The resulting p-values are reported in Tables 5.8 and 5.9. Additionally, the behavior of thaw-variants is shown in Figures 5.19 through 5.24. In these figures, α and validation accuracy are plotted throughout the training process. The α values are frozen until the 22nd epoch, as explained by table 4.3.

The addition of a thawing factor does not show improvements compared with either MH or HBP variants. Figures 5.19 through 5.24 show that while the approach of boundary values is delayed, the boundaries are still met consistently, meaning that the addition of a thawing period does not solve the stability issues of HBP.

Table 5.8: Comparison of mean performance across all data splits and folds between MHResNet34 and HBPResNet34, and ThawResNet34. A pairwise t-test is used to determine a significant difference in means, where difference is defined as mean of ThawResNet34 - mean of MHResNet34 or HBPResNet34. A positive difference means ThawResNet34 outperforms MHResnet34 or HBPResNet34, and vice versa. The resulting p-value relates to the null hypothesis that MHResNet34 or HBPResNet34 are not significantly different to ThawResNet34.

	MH		HBP	
Experiment	p-value	Diff. in means	p-value	Diff. in means
All	0.0178	0.61	<0.0001	4.11
CIFAR10	0.923	-0.024	0.274	0.368
Papilionidae, untrained	0.068	1.288	0.2116	0.336
Papilionidae, pretrained	0.012	0.580	0.0014	1.396

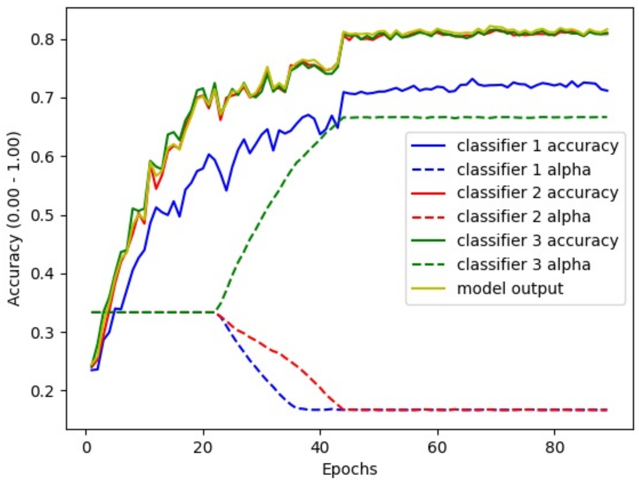

Figure 5.19: Results for ThawResNet34 for individual classifiers on 20% of training data for CIFAR10. Shown is the validation accuracy over all training epochs for each individual classifier and the combined weighted output.

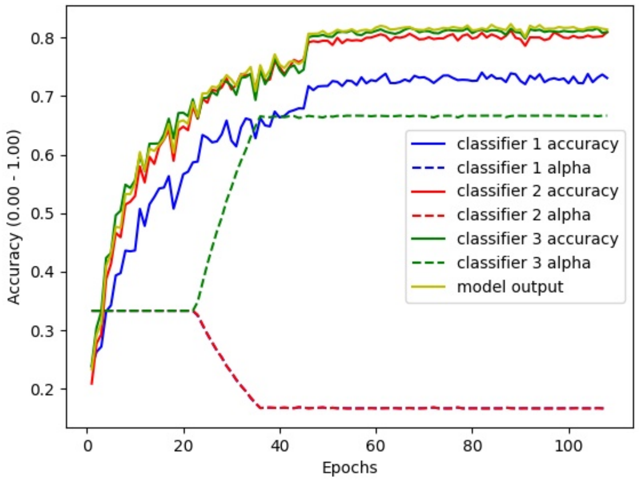

Figure 5.20: Results for ThawDenseNet121 for individual classifiers on 20% of training data for CIFAR10. Shown is the validation accuracy over all training epochs for each individual classifier and the combined weighted output.

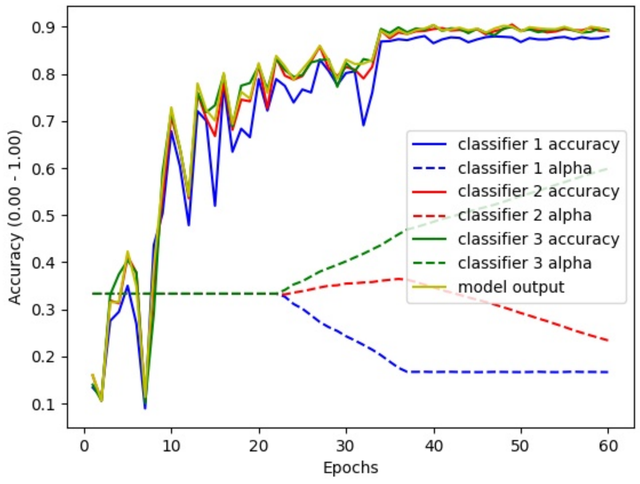

Figure 5.21: Results for ThawResNet34 for individual classifiers on 20% of training data for Papilionidae, untrained. Shown is the validation accuracy over all training epochs for each individual classifier and the combined weighted output.

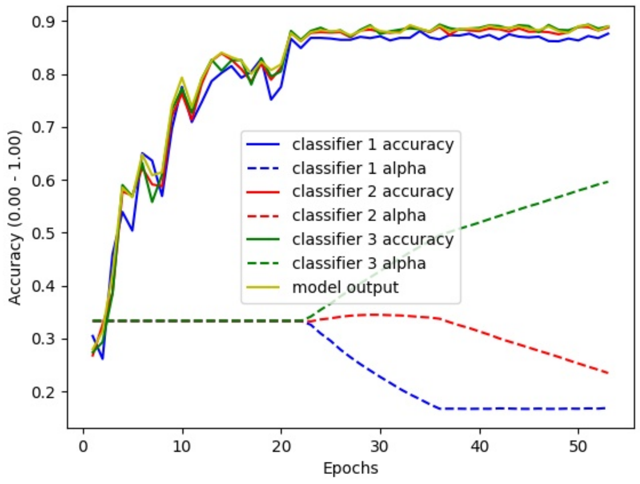

Figure 5.22: Results for ThawDenseNet121 for individual classifiers on 20% of training data for Papilionidae, untrained. Shown is the validation accuracy over all training epochs for each individual classifier and the combined weighted output.

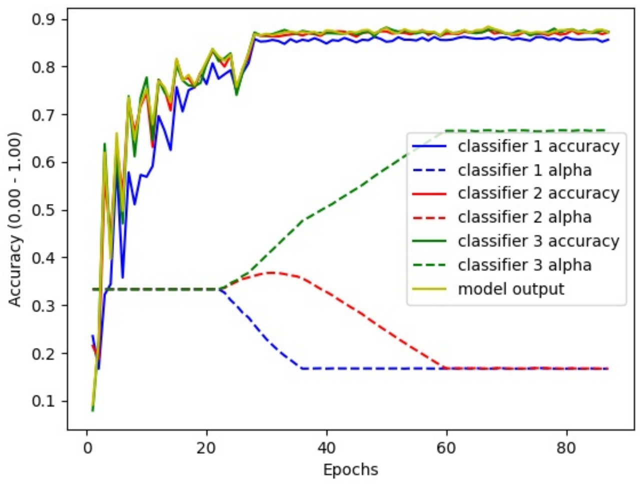

Figure 5.23: Results for ThawResNet34 for individual classifiers on 20% of training data for Papilionidae, pretrained. Shown is the validation accuracy over all training epochs for each individual classifier and the combined weighted output.

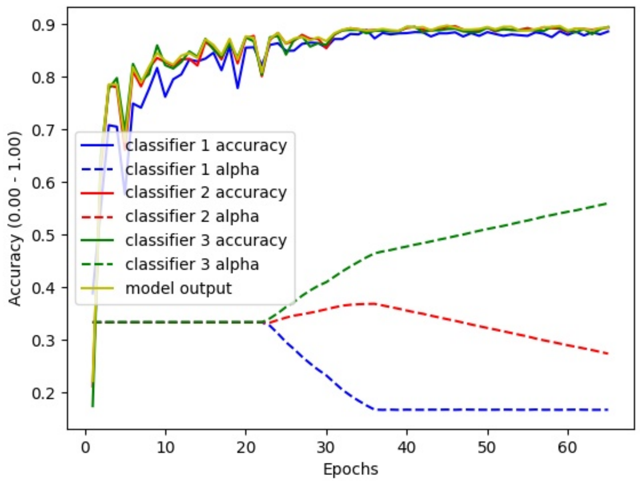

Figure 5.24: Results for ThawDenseNet121 for individual classifiers on 20% of training data for Papilionidae, pretrained. Shown is the validation accuracy over all training epochs for each individual classifier and the combined weighted output.

Table 5.9: Comparison of mean performance across all data splits and folds between MHDenseNet121 and HBPDenseNet121, and ThawDenseNet121. A pairwise t-test is used to determine a significant difference in means, where difference is defined as mean of ThawDenseNet121 - mean of MHDenseNet121 or HBPDenseNet121. A positive difference means ThawDenseNet121 outperforms MHDenseNet121 or HBPDenseNet121, and vice versa. The resulting p-value relates to the null hypothesis that MHDenseNet121 or HBPDenseNet121 are not significantly different to ThawDenseNet121.

	MH		Thaw	
Experiment	p-value	Diff. in means	p-value	Diff. in means
All	0.515	-0.104	0.877	-0.023
CIFAR10	0.282	-0.384	0.756	0.104
Papilionidae, untrained	0.763	0.092	0.0617	0.468
Papilionidae, pretrained	0.874	-0.020	0.0392	0.296

5.5 Do the behaviours of these models change when using differing amounts of data?

Figures 5.25 through 5.27 show the performance of each model with increasing data for each of the experimental settings. From these figures, it is clear that the relative performance of each model does not change significantly when increasing the amount of training data.

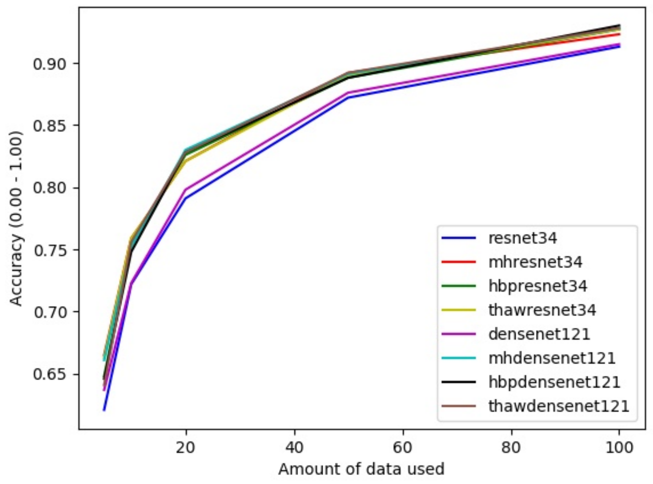

Figure 5.25: Mean results for all models on all folds for CIFAR10, plotted over the amount of data used.

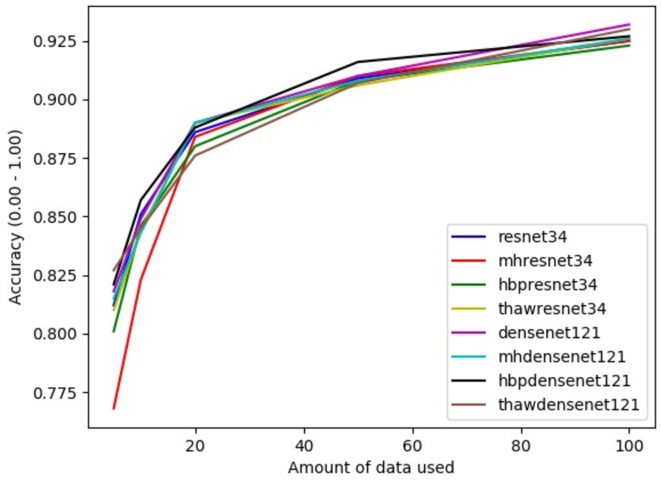

Figure 5.26: Mean results for all models on all folds for Papilionidae, untrained, plotted over the amount of data used.

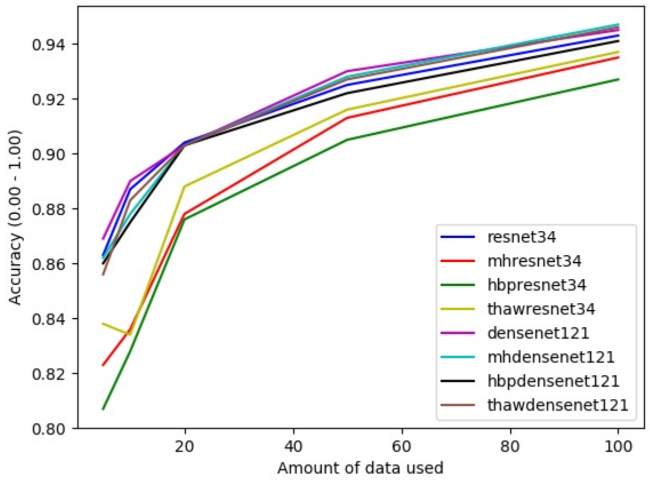

Figure 5.27: Mean results for all models on all folds for Papilionidae, pretrained, plotted over the amount of data used.

Chapter 6

Discussion

The results, as shown in Chapter 5, show several patterns. However, these patterns seem to be experiment specific, with the analysis on CIFAR10 differing significantly from experiments on Papilionidae, both with and without transfer learning. Therefore, when applicable, the research questions will be answered with emphasis on individual experiments, rather than with more generalizable statements. The results from Chapter 5 are summarized below, before being further discussed.

6.1 Research Questions

Does a model with multiple heads perform better than the base model?

From Tables 5.4 and 5.5, it shows that MH variants outperform base variants only on CIFAR10. The follow-up question of why this difference between experiments occurs leads to Figures 5.1 to 5.6, which shows the effects of the MH variant on regularizing the training process, providing a more generalizable model. The MH variants are able to optimize better due to a more gradual decrease in the loss and validation accuracy, preventing the early stopping mechanism from intervening too aggresively. These effects of more gradual learning and subsequent performance increase for MH variants do not occur in experiments on the Papilionidae data, both with and without transfer learning. In conclusion, the use of MH variants may increase performance on classification tasks using CNNs compared to a base version. However, this gain varies between tasks.

Do the additional heads increase performance when active during inference, in contrast to only using additional heads during training?

From Figures 5.7 and 5.8, it shows that the combined weighted output of the MH model variants slightly outperforms the third, final classifier. In established deep supervision techniques, the early classifiers are removed at inference time, which is similar to looking at the performance of classifier 3 in the MH variant. Because the combined output outperforms the third classifier in the CIFAR10 experimental setting, it shows that leaving the additional classifiers active during inference provides a benefit. This effect was not observed for other experimental settings, as for these the MH variant did not outperform the base variant. There was no difference in performance between the third classifier and the combined output in the experiments with the Papilionidae dataset, both with and without transfer learning.

Does the hedge backpropagation (HBP) algorithm successfully optimize weights between output layers of a model with multiple heads?

Tables 5.6 and 5.7 show the difference in performance between MH and HBP variants. From these tables it is clear that the addition of HBP does not improve the training of the architecture. The differences in performance are either statistically insignificant, negligibly small, or in favor of the MH variant. Additionally, Figures 5.13 through 5.18 show the development of α in each of the experiments. While the development of α throughout training is different based on the model and the data used, the end result is that the final classifier reaches the upper bound for α, while all other classifiers reach the lower bound. This indicates that α is unstable and the HBP algorithm is sub-optimal.

Does freezing the weights assigned to classifiers in the model at the start of training influence the performance of the model? Does this significantly differ from the MH or HBP models?

In Tables 5.8 and 5.9, the performance of the Thaw variants is compared to both MH and HBP variants. Similar to the comparison between MH and HBP variants, these tables show that any discernible difference in the performance of these variants is either statistically insignificant or negligibly small. Figures 5.19 through 5.24 show the development of α in Thaw variants for all settings. From these results it shows that the final performance of the model does not increase compared to either MH or HBP variants for any of the experimental settings. The effect of freezing alpha development for 22 epochs seems to have mainly a delaying effect. The α values for the classifiers do not reach their maximum and minimum value as quickly and in some scenarios the upper and lower bounds are not reached at all. However, the curve does not seem to flatten yet, indicating that the only reason for the bounds not being reached is early stopping, rather than a stable distribution for α being reached.

Do the behaviours of these models change when using differing amounts of data?

Shown in Figures 5.25 through 5.27 are the increases in performance per model when training data is increased. From these figures it shows that the relative performance of different model variants does not change considerably with different amounts of data. In other words, when looking at Figure 5.25, the base variant consistently performs worse than variants with multiple heads, regardless of amount of training data used. Similarly, there is no difference in relative performance between variants in Figures 5.26 and 5.27. This shows no indication that using additional classifiers in shallow parts of the model allows the model to adapt to the unavailability of data by focusing on the optimization of a shallower model.

6.2 Conclusions

This work shows the benefits of the Multiple Heads architecture in some, but not all, scenarios. The established technique of deep supervision has already been shown to improve training of certain models, and that is verified in this work. However, an additional hypothesis arose that in some of the circumstances where deep supervision is beneficial, keeping the additional classifiers intact at inference time may increase model performance to some extent. With the data provided by experiments in this work, no conclusive statements can be made about general applicability, but specifically for the CIFAR10 setting a significant benefit has been observed. The fact that this observation is seen in both ResNet34 and DenseNet121 puts more weight behind it.

Unfortunately, this is where the benefits of this work stagnate. The exploration of a new algorithm to determine weights between heads in the MH model variant has been unsuccessful. Both HBP and Thaw variants show no benefits on top off the benefits from the MH variant. Interesting to note is that the variants using an adaptable α also do not appear to hinder the performance. This is likely due to setting rigorous upper and lower bounds for α, as all figures show that the bounds are consistently reached, albeit with a delay in the case of the Thaw variants. This indicates a highly unstable system. Initially the HBP algorithm was developed not for convolutional neural networks (CNNs), but for Multi-Layer Perceptrons (MLPs) of significant depth [Sahoo et al., 2018]. Additionally, this was done in the interest of online deep learning, assuming each sample would only be viewed once. While it is unclear whether these circumstances prevent α from reaching its upper and lower bounds, it is intuitive to see that continuous iteration over the same data will push α to extremes as soon as a clear best classifier establishes itself. The adaptation to CNNs was based on the idea that earlier classifiers would be more dominant early on in training, and only diminish in value after a period of time when the final classifier learns all the details of the data. However, the empirical results have shown that the final classifier is able to outperform other classifiers from the start, resulting in the α for this classifier increasing, its training picking up more speed, which further feeds into its superiority over the other classifiers. This becomes a feedback loop where the final classifier will be dominant throughout all stages of the training process.

If the final classifier outperforms earlier classifiers, why is the MH variant then able to outperform a base variant? What do the earlier classifiers add to improve the architecture? The additional classifiers appear to help in regularizing the network. This is seen in the convergence behaviour for experiments on CIFAR10, where the more gradual loss curve for MH variants allows the model to surpass the base variant without interference from the early stopping mechanism. In other words, the behaviour indicates that the MH adaptation allows the model to avoid a local minimum that the base variant is more prone to get stuck in.

Addtionally, while both ResNets and DenseNets have their own structures to reduce the effects of the vanishing gradient problem, the fact remains that details from earlier layers will get lost in the final classification. The addition of earlier classifiers, as in deep supervision, helps by giving the earlier convolutional blocks more direct feedback on the features being extracted. As the experimental data shows, for the models tested in this work, and with specific datasets, this helps to keep the model more generalized at inference time.

Finally, the original work, being developed for online learning, assumed that a system would gradually see more data and adapt to that by using the deeper classifiers. As such, this work posited the question of whether the behavioural comparison between different model variants would change given the amount of data used. While this is not the same as online learning, the question is relevant in modern deep learning research due to improvements often being contingent on very large datasets. Having an architecture that would also function with lower amounts of data, or outperform the base net by a larger margin compared to using a bigger dataset, can be very beneficial. However, the empirical results show that the relative performance of the models does not differ very much even with an increase in data by a factor of 20 (5% data compared to 100% data). The performance gap closes somewhat, but that is also attributable to both base and MH variants achieving accuracies closer to 100. The reduction in error remains similar for increasing amounts of data.

6.3 Recommendations for Future Research

The steps towards future research encompass two directions. The first is to determine the validity of the statements in this work by expanding the field of view. This means the models developed herein would be applied to numerous datasets in an attempt to detect a pattern in the type of data these models can help with or not. To broaden the field of view, a direct comparison can also be made between the models developed in this work and other works using auxiliary heads for deep supervision. These comparisons can focus on the added value of the auxiliary heads both at training and inference time. Examples of this include GoogLeNet [Szegedy et al., 2015] and Inception v3 [Szegedy et al., 2016].

Future work may also focus on expanding the techniques developed in this work in order to address the issues inherent to HBP. This can be done by, for example, developing a model that fully trains the weights until convergence before adapting α. This allows each classifier to find its individual optimal contribution before determining the amount each classifier contributes to the weighted output. One step further would be to take this two-step training process and develop en n-step training process where weight adaptation and alpha adaptation are alternated until some convergence criteria is met.

Alternatively, an architecture can be developed that functions as the reverse of the Thaw variant. In this architecture, the alpha would be adaptable until a certain epoch, basically prohibiting the alpha from always reaching its boundary value by bounding it in the time domain. And a combination of thawing and freezing α could result in a model that has a static α at the start of training, adapts it after a preset n number of epochs before freezing α again at epoch m, where $m > n$. These adaptations, however, attempt to shape α in a certain way, and the choices and additional hyper parameters that this entails would very closely start to resemble a MH variant with predetermined, possibly non-equal, weights. These variants do not solve the inherent issues with HBP in that it does not converge any other way than by reaching the preset upper and lower bounds.

Theoretically, the HBP variants should outperform MH variants on the simple basis that if the optimal weighted combination of classifiers is found, these should either be better than a statically weighted ensemble or converge on the same weights as the statically weighted ensemble. This leads to the idea that this can be achieved if a stable method of updating α can be found. A start may be to use the log loss instead of the real loss, such that the discrepancy between large losses at the start and small losses at the end of training are smoothed. This difference in size of the loss directly affects α calculation and smoothing it may balance impact of α updates from early and late in the training process. Of course, further algorithms can be developed which adapt α in an online fashion, and these heavily influence the comparison between HBP and MH variants if applied, but developing these is beyond the scope of this work.

Finally, an interesting point for future research is to revisit two of the decisions made early in development. The first of these is the choice to use small versions of both ResNet and DenseNet. It is argued that this would limit the space between classifiers, i.e. by limiting the convolutional layers per block. However, during development and testing it became clear that the opposite could be equally interesting. By using a network with large blocks, in a similar block structure to the architectures in this work, the classifiers are each responsible for updating a larger portion of the network. This would increase the severity of the vanishing gradient problem and the disconnect between early layers and the final classification. As such, in these deeper, harder to train networks, the impact of additional classifiers might be more noticeable than in the relatively shallow networks used in this work.

Second, there is the decision to limit the architecture to 3 classifiers. This decision was partly based on computational limitations and partly on early observations during development. These early observations pointed in the direction that classifiers attached to convolutional layers before the point where the first classifier is currently attached would only hinder performance by confusing the purpose of these early layers. However, in combination with the previous point, when using larger convolutional blocks, resulting in larger amounts of encoder layers, classifiers placed between the currently used classifiers may add additional value without confusing early layers. To do this, the block structure as described in Section 3.2 has to let go to some extent, but it could be interesting to explore this nonetheless.

In closing: The pessimist might say that the architectures developed here are of little value, adding minor performance in a limited set of circumstances. The optimist might say that gain has been found and the search for a new and better architecture was a success. The realist, however, will say that neither the good nor the bad is conclusive and there is much room for further work.

Bibliography

[Dietterich, 2000] Dietterich, T. G. (2000). Ensemble methods in machine learning. In *Multiple Classifier Systems*, pages 1–15, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Freund and Schapire, 1997] Freund, Y. and Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119 – 139.

[Gertych et al., 2019] Gertych, A., Swiderska-Chadaj, Z., Ma, Z., Ing, N., Markiewicz, T., Cierniak, S., Salemi, H., Guzman, S., Walts, A., and Knudsen, B. (2019). Convolutional neural networks can accurately distinguish four histologic growth patterns of lung adenocarcinoma in digital slides. *Scientific Reports*, 9.

[He et al., 2016] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778.

[Hochreiter, 1991] Hochreiter, S. (1991). Untersuchungen zu dynamischen neuronalen netzen.

[Huang et al., 2017a] Huang, G., Li, Y., Pleiss, G., Liu, Z., Hopcroft, J. E., and Weinberger, K. Q. (2017a). Snapshot ensembles: Train 1, get m for free. *ArXiv*, abs/1704.00109.

[Huang et al., 2017b] Huang, G., Liu, Z., and Weinberger, K. Q. (2017b). Densely connected convolutional networks. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2261–2269.

[Kolen and Kremer, 2001] Kolen, J. F. and Kremer, S. C. (2001). Gradient flow in recurrent nets: The difficulty of learning longterm dependencies. In *A Field Guide to Dynamical Recurrent Networks*, pages 237–243.

[Krizhevsky, 2009] Krizhevsky, A. (2009). Learning multiple layers of features from tiny images. *University of Toronto*, pages 32–33.

[Krizhevsky et al., 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Pereira, F., Burges, C. J. C., Bottou, L., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc.

[Lecun et al., 1998] Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

[Lee et al., 2015] Lee, C.-Y., Xie, S., Gallagher, P., Zhang, Z., and Tu, Z. (2015). Deeply-Supervised Nets. In *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics*, volume 38 of *Proceedings of Machine Learning Research*, pages 562–570.

[Li et al., 2019] Li, C., Zia, M. Z., Tran, Q.-H., Yu, X., Hager, G., and Chandraker, M. (2019). Deep supervision with intermediate concepts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41:1828–1843.

[Marre et al., 2020] Marre, G., De Almeida Braga, C., Ienco, D., Luque, S., Holon, F., and Deter, J. (2020). Deep convolutional neural networks to monitor coralligenous reefs: Operationalizing biodiversity and ecological assessment. *Ecological Informatics*, 59:101110.

[Minetto et al., 2019] Minetto, R., Segundo, M. P., and Sarkar, S. (2019). Hydra: An ensemble of convolutional neural networks for geospatial land classification. *IEEE Transactions on Geoscience and Remote Sensing*, 57:6530–6541.

[Naturalis Biodiversity Center, 2021] Naturalis Biodiversity Center (2021). Papilionidae specimens.

[Paszke et al., 2019] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

[Rosenblatt, 1958] Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65 6:386–408.

[Rumelhart et al., 1986] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning Representations by Back-propagating Errors. *Nature*, 323(6088):533–536.

[Sahoo et al., 2018] Sahoo, D., Pham, Q., Lu, J., and Hoi, S. C. H. (2018). Online deep learning: Learning deep neural networks on the fly. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pages 2660–2666. International Joint Conferences on Artificial Intelligence Organization.

[Shen et al., 2020] Shen, Z., Liu, Z., Li, J., Jiang, Y. G., Chen, Y., and Xue, X. (2020). Object detection from scratch with deep supervision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(2):398–412.

[Sigcha et al., 2021] Sigcha, L. F., Pavón, I., Costa, N., Costa, S., Gago, M., Arezes, P., Lopez Navarro, J. M., and Arcas, G. (2021). Automatic resting tremor assessment in parkinson's disease using smartwatches and multitask convolutional neural networks. *Sensors*, 21:291.

[Simonyan and Zisserman, 2015] Simonyan, K. and Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. In Bengio, Y. and LeCun, Y., editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

[Sun et al., 2019] Sun, D., Yao, A., Zhou, A., and Zhao, H. (2019). Deeply-supervised knowledge synergy. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6990–6999.

[Szegedy et al., 2015] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going deeper with convolutions. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9.

[Szegedy et al., 2016] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. (2016). Rethinking the inception architecture for computer vision. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 2818–2826. IEEE Computer Society.

[Wang et al., 2015] Wang, L., Lee, C.-Y., Tu, Z., and Lazebnik, S. (2015). Training deeper convolutional networks with deep supervision. *ArXiv*, abs/1505.02496.

[Xu et al., 2015] Xu, B., Wang, N., Chen, T., and Li, M. (2015). Empirical evaluation of rectified activations in convolutional network. *CoRR*, abs/1505.00853.

[Ye et al., 2017] Ye, C., Yang, Y., Mao, R., Fermüller, C., and Aloimonos, Y. (2017). What can i do around here? deep functional scene understanding for cognitive robots. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4604–4611.

[Yutong and Zhao, 2015] Yutong, S. and Zhao, H. (2015). Stock selection model based on advanced adaboost algorithm. In *2015 7th International Conference on Modelling, Identification and Control (ICMIC)*, pages 1–7.

[Zhou et al., 2002] Zhou, Z.-H., Wu, J., and Tang, W. (2002). Ensembling neural networks: Many could be better than all. *Artificial Intelligence*, 137(1):239–263.

Chapter 7

Appendix A - Full Results

Table 7.1: CIFAR10: Validation accuracy for all models on the CIFAR10 dataset using 5% of training data. Best performing variant per base model is shown in bold.

Model	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean
ResNet34	61.3	60.6	62.5	62.9	63.3	62.1
MHResNet34	68.4	64.6	67.3	65.6	66.8	**66.5**
HBPResNet34	64.4	61.8	66.8	66.6	64.3	64.8
ThawResNet34	67.7	63.8	65.9	64.0	69.9	66.3
Densenet121	63.0	62.9	63.4	63.1	65.9	63.7
MHDenseNet121	66.7	67.0	67.6	63.8	65.6	**66.1**
HBPDenseNet121	67.2	65.4	63.5	64.5	62.6	64.6
ThawDenseNet121	62.4	64.2	62.8	66.9	64.3	64.1

Table 7.2: CIFAR10: Validation accuracy for all models on the CIFAR10 dataset using 10% of training data. Best performing variant per base model is shown in bold.

Model	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean
ResNet34	72.6	71.8	70.7	72.6	73.3	72.2
MHResNet34	75.5	75.5	75.6	76.3	76.7	**75.9**
HBPResNet34	75.5	73.9	75.2	77.1	74.6	75.3
ThawResNet34	77.5	74.0	76.3	75.8	75.9	**75.9**
Densenet121	73.1	69.0	73.5	72.6	73.4	72.3
MHDenseNet121	76.9	74.8	75.5	75.7	73.2	75.2
HBPDenseNet121	74.8	74.3	72.2	76.7	75.9	74.8
ThawDenseNet121	76.5	75.0	76.1	74.4	76.2	**75.6**

Table 7.3: CIFAR10: Validation accuracy for all models on the CIFAR10 dataset using 20% of training data. Best performing variant per base model is shown in bold.

Model	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean
ResNet34	79.8	78.2	79.0	79.5	79.1	79.1
MHResNet34	82.1	80.7	82.1	82.5	83.2	82.1
HBPResNet34	83.1	81.9	82.8	83.2	82.2	**82.6**
ThawResNet34	81.7	82.6	82.2	82.0	82.2	82.1
Densenet121	78.8	79.5	79.4	80.9	80.2	79.8
MHDenseNet121	83.0	82.7	83.2	83.5	82.7	**83.0**
HBPDenseNet121	83.7	82.4	82.6	83.1	82.2	82.8
ThawDenseNet121	83.1	82.3	82.8	83.6	82.3	82.8

Table 7.4: CIFAR10: Validation accuracy for all models on the CIFAR10 dataset using 50% of training data. Best performing variant per base model is shown in bold.

Model	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean
ResNet34	87.2	87.6	87.1	86.9	87.4	87.2
MHResNet34	89.6	89.0	89.8	89.4	88.3	**89.2**
HBPResNet34	88.4	89.2	89.3	87.8	89.3	88.8
ThawResNet34	88.3	88.7	89.6	88.2	90.2	89.0
Densenet121	86.7	87.5	88.2	87.6	88.1	87.6
MHDenseNet121	88.9	89.2	89.8	88.4	89.2	89.1
HBPDenseNet121	89.0	88.0	90.2	88.8	88.1	88.8
ThawDenseNet121	89.4	89.8	89.0	88.3	89.3	**89.2**

Table 7.5: CIFAR10: Validation accuracy for all models on the CIFAR10 dataset using 100% of training data. Best performing variant per base model is shown in bold.

Model	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean
ResNet34	91.4	90.9	91.6	91.2	91.2	91.3
MHResNet34	92.5	92.5	91.8	92.4	92.4	92.3
HBPResNet34	92.6	93.0	93.1	92.5	92.2	**92.7**
ThawResNet34	92.7	92.7	92.9	92.9	92.3	**92.7**
Densenet121	91.2	91.6	91.5	91.7	91.7	91.5
MHDenseNet121	92.7	92.8	93.0	92.6	92.9	92.8
HBPDenseNet121	92.9	93.0	93.2	92.8	93.1	**93.0**
ThawDenseNet121	93.5	92.7	92.9	92.8	92.2	92.8

Table 7.6: Validation accuracy for all models on the Papilionidae dataset using 5% of training data. Best performing variant per base model is shown in bold.

Model	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean
ResNet34	83.7	82.2	78.4	82.5	79.2	**81.2**
MHResNet34	82.1	76.0	69.8	72.9	83.3	76.8
HBPResNet34	81.6	76.9	80.0	81.4	80.8	80.1
ThawResNet34	84.7	78.0	78.2	83.5	80.7	81.0
Densenet121	83.8	80.7	79.4	81.7	83.2	81.8
MHDenseNet121	82.5	82.7	81.7	80.7	79.9	81.5
HBPDenseNet121	82.6	83.5	81.8	81.9	80.5	82.1
ThawDenseNet121	84.1	81.8	82.8	83.6	81.1	**82.7**

Table 7.7: Validation accuracy for all models on the Papilionidae dataset using 10% of training data. Best performing variant per base model is shown in bold.

Model	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean
ResNet34	87.1	84.9	85.6	85.2	82.5	**85.1**
MHResNet34	86.6	83.5	72.6	84.7	84.1	82.3
HBPResNet34	87.0	83.8	83.5	84.6	84.3	84.6
ThawResNet34	87.0	81.5	82.6	86.1	84.6	84.4
Densenet121	87.8	83.1	85.3	84.7	83.5	84.9
MHDenseNet121	87.7	84.1	80.4	83.6	85.5	84.3
HBPDenseNet121	87.3	84.6	85.7	87.4	83.6	**85.7**
ThawDenseNet121	86.5	82.4	84.0	85.5	84.3	84.5

Table 7.8: Validation accuracy for all models on the Papilionidae dataset using 20% of training data. Best performing variant per base model is shown in bold.

Model	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean
ResNet34	91.3	87.7	87.2	89.4	87.6	88.6
MHResNet34	90.3	89.0	86.7	88.7	87.5	88.4
HBPResNet34	89.8	86.3	87.8	89.5	86.7	88.0
ThawResNet34	90.3	89.2	88.2	89.4	87.9	**89.0**
Densenet121	90.8	88.4	87.1	89.8	88.7	**89.0**
MHDenseNet121	90.6	89.6	87.2	89.3	88.1	**89.0**
HBPDenseNet121	90.8	87.6	88.0	89.9	87.5	88.8
ThawDenseNet121	89.2	87.0	85.7	88.3	88.0	87.6

Table 7.9: Validation accuracy for all models on the Papilionidae dataset using 50% of training data. Best performing variant per base model is shown in bold.

Model	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean
ResNet34	92.8	89.6	91.5	91.3	89.1	90.9
MHResNet34	92.9	89.5	90.7	92.4	89.7	**91.0**
HBPResNet34	92.5	91.1	89.3	91.9	89.0	90.8
ThawResNet34	92.7	89.2	89.8	91.6	89.5	90.6
Densenet121	92.5	90.1	89.9	91.7	90.8	91.0
MHDenseNet121	92.5	90.9	89.8	91.7	89.0	90.8
HBPDenseNet121	92.7	91.1	91.2	92.8	90.3	**91.6**
ThawDenseNet121	92.4	89.0	90.4	92.2	89.6	90.7

Table 7.10: Validation accuracy for all models on the Papilionidae dataset using 100% of training data. Best performing variant per base model is shown in bold.

Model	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean
ResNet34	93.2	92.7	91.9	92.7	91.9	92.5
MHResNet34	92.9	92.1	92.4	93.8	91.1	92.5
HBPResNet34	92.7	92.5	91.7	93.3	91.1	92.3
ThawResNet34	93.4	92.3	93.0	92.8	91.3	**92.6**
Densenet121	93.8	92.9	93.4	93.6	92.3	**93.2**
MHDenseNet121	93.7	92.3	92.7	93.6	90.9	92.6
HBPDenseNet121	93.4	92.0	93.2	93.6	91.7	92.7
ThawDenseNet121	93.7	92.7	92.9	93.8	92.0	93.0

Table 7.11: Validation accuracy for all models, pretrained on ImageNet, on the Papilionidae dataset using 5% of training data. Best performing variant per base model is shown in bold.

Model	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean
ResNet34	87.7	86.1	84.5	87.2	86.0	**86.3**
MHResNet34	85.3	80.6	79.3	83.0	83.1	82.3
HBPResNet34	82.7	80.7	81.7	78.0	80.3	80.7
ThawResNet34	85.4	84.4	80.8	84.2	84.4	83.8
Densenet121	87.3	86.0	85.5	88.6	87.1	**86.9**
MHDenseNet121	87.4	85.0	84.6	87.7	86.4	86.2
HBPDenseNet121	87.1	85.3	84.7	87.2	85.6	86.0
ThawDenseNet121	86.5	85.0	84.4	86.4	85.6	85.6

Table 7.12: Validation accuracy for all models, pretrained on ImageNet, on the Papilionidae dataset using 10% of training data. Best performing variant per base model is shown in bold.

Model	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean
ResNet34	90.5	88.3	87.4	89.6	87.6	**88.7**
MHResNet34	87.4	81.4	81.5	84.3	83.5	83.6
HBPResNet34	86.9	83.4	83.9	81.1	78.9	82.8
ThawResNet34	87.3	81.8	80.8	84.8	82.4	83.4
Densenet121	90.0	87.8	88.4	90.6	88.1	**89.0**
MHDenseNet121	89.6	86.6	86.6	89.2	86.8	87.8
HBPDenseNet121	89.6	86.0	86.9	87.9	87.1	87.5
ThawDenseNet121	90.5	87.6	86.3	89.6	87.7	88.3

Table 7.13: Validation accuracy for all models, pretrained on ImageNet, on the Papilionidae dataset using 20% of training data. Best performing variant per base model is shown in bold.

Model	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean
ResNet34	92.0	90.2	89.5	90.7	89.6	**90.4**
MHResNet34	90.1	86.9	84.4	90.6	86.9	87.8
HBPResNet34	87.9	88.0	85.6	89.3	87.1	87.6
ThawResNet34	90.0	88.4	87.3	90.8	87.7	88.8
Densenet121	92.3	89.6	89.2	91.3	89.3	**90.3**
MHDenseNet121	92.5	89.6	88.6	91.2	89.5	**90.3**
HBPDenseNet121	92.4	90.1	89.3	90.9	88.9	**90.3**
ThawDenseNet121	92.1	89.7	89.2	91.3	89.2	**90.3**

Table 7.14: Validation accuracy for all models, pretrained on ImageNet, on the Papilionidae dataset using 50% of training data. Best performing variant per base model is shown in bold.

Model	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean
ResNet34	94.2	92.1	92.1	92.3	91.9	**92.5**
MHResNet34	93.0	90.7	90.1	92.0	90.5	91.3
HBPResNet34	92.3	89.7	89.1	92.1	89.5	90.5
ThawResNet34	92.9	91.2	90.7	92.6	90.5	91.6
Densenet121	94.1	92.7	92.9	93.4	92.1	93.0
MHDenseNet121	93.4	92.7	91.9	93.4	92.4	92.8
HBPDenseNet121	93.6	91.8	91.7	93.2	90.7	92.2
ThawDenseNet121	93.8	92.4	92.1	93.5	91.7	92.7

Table 7.15: Validation accuracy for all models, pretrained on ImageNet, on the Papilionidae dataset using 100% of training data. Best performing variant per base model is shown in bold.

Model	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean
ResNet34	94.6	94.2	94.5	94.9	93.2	**94.3**
MHResNet34	94.0	93.4	93.0	94.6	92.7	93.5
HBPResNet34	93.6	91.0	93.3	93.7	92.1	92.7
ThawResNet34	94.1	93.1	94.1	94.5	92.6	93.7
Densenet121	95.1	93.8	94.5	95.1	94.0	94.5
MHDenseNet121	95.0	94.8	94.2	95.3	94.1	**94.7**
HBPDenseNet121	94.5	94.5	94.0	94.3	93.3	94.1
ThawDenseNet121	94.2	94.9	94.7	95.0	94.2	94.6