MASTER THESIS

Progressively Growing Neural Networks for Scene Graph Generation from Images

Author:
Viviane KAKERBECK

Supervisor:
Pascal NIETERS
Second Supervisor:
Peter KÖNIG

A thesis submitted in fulfillment of the requirements for the degree of Master of Science

in the

Neuroinformatics Group
Institute of Cognitive Science

October 23, 2018

Declaration of Authorship

I hereby certify that the work presented here is, to the best of my knowledge and belief, original and the result of my own investigations, except as acknowledged, and has not been submitted, either in part or whole, for a degree at this or any other university.

_____

signature

_____

city, date

University of Osnabrück

Abstract

Institute of Cognitive Science

Master of Science

Progressively Growing Neural Networks for Scene Graph Generation from Images

by Viviane KAKERBECK

In this thesis I will explore the possibility of progressively learning new information during neural network training by using a growing network which is increasing in size according to the complexity of the task assigned to it. Just as humans can keep learning during life this ability can be very helpful in many applications. So far however, neural networks struggle with catastrophic forgetting meaning the network suffers a complete loss of old information once a new task is introduced. I do not solve this problem but rather find a way to maneuver around it in a manner that can be applied when the new task has the same properties as the old task such as both tasks being trained simultaneously. This will be exemplified through image recognition with the help of convolutional neural networks and tested on three different data sets of varying sizes. The results indicate that a network can learn new object categories on the fly while still performing well on the previously learned categories. Additionally, I show how this method of continuous learning can even exceed the performance of a conventionally trained network when scheduled in an order that increases in difficulty. Further exploration of different methods of combating catastrophic forgetting such as freezing network layers or weighted class training has not led to better results. Overall the method of continuous learning and network growing combined with old class rehearsal will be shown to be equal, in some cases better, than conventional network training. Confirming the use of this method in real world applications, opening up many new and practical possibilities.

Contents

1 Introduction

1.1 Task Definition and Problem Refinement

The long-term goal of this project is to implement a system that can detect a large number of objects in images and their relationships with each other. This system should be able to learn with the help of user input and thereby improve its accuracy and its knowledge about the world. The network architecture could look something like what is shown in figure 1.1.

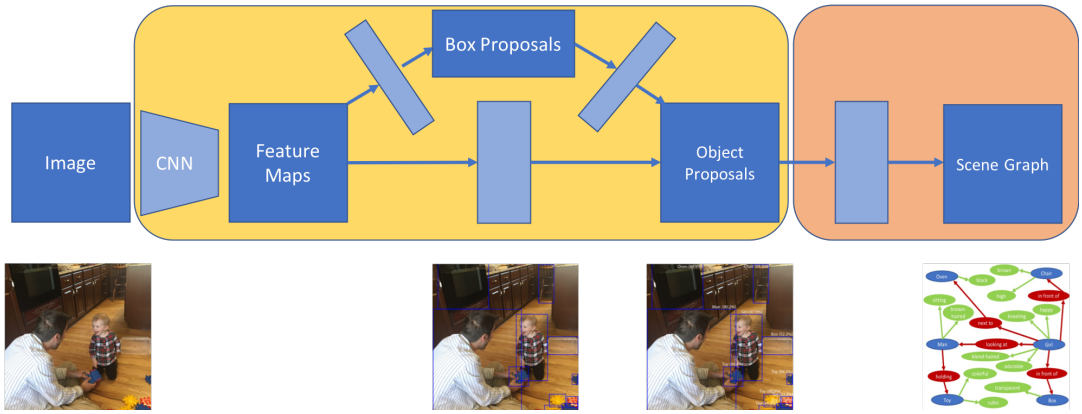

FIGURE 1.1: Overall network architecture for extracting scene graphs from images. Light blue boxes represent neural networks with several layers. Dark blue boxes represent (intermediate) network outputs. Everything in the yellow box will be trained in one session. The output of this will be fed into a separately trained network

When implementing this architecture several problems arise. This thesis will cover the problem of progressively learning new classes during the training of a neural network without forgetting the old classes. It will investigate how networks react to a new class in the data set, if the network can grow over time, and the effects on big and small data sets. Additionally, I will further investigate catastrophic forgetting as well as curriculum learning which describes the order in which classes are learned by a network.

1.2 Related Work

1.2.1 Object Recognition and Scene Segmentation

There have been several attempts made to extract more information from an image than a simple object label. This ranges from recognition of a scene composed of different objects (eg: horse + man + ball = polo) (Li, Socher, and Fei-Fei, 2009) to the generation of detailed scene graphs expressing attributes and relationships of objects in an image (Xu et al., 2017). There are several different pieces of information that can be extracted from images to learn about their content.

One option is *object detection* in which the location of objects in images is detected. This can be visualized as bounding boxes around distinct objects in a scene, such as people or faces (Viola and Jones, 2001). Labeling training data with bounding boxes is relatively easy and can even be done during a real-time training process of an object detection network (Teng, Huang, and Iannucci, 2018).

A more complex approach is *object or scene segmentation* in which we don't just detect the location and size of an object but also the exact shape of it (Dvornik et al., 2017). The resulting segmentation should then give the outline of specific objects or the border between the foreground and background. Especially good results for segmenting unknown objects can be archived by using RGB-D images (Asif, Bennamoun, and Sohel, 2016).

The aforementioned approaches can then be combined with *object recognition*. In this case not only are the shape and location of an object detected but also the object itself is named. By combining object detection and object recognition we can now label many objects in one image (Johnson, Karpathy, and Fei-Fei, 2015) and even describe relationships between objects in meaningful sentences (Karpathy, Joulin, and Fei-Fei, 2014).

The next step is *scene understanding* in which we try to extract the overall semantic meaning of an image. Here we look at the composition of all objects in an image and the potential interaction between them. For example, we can then extract labels like "tennis game" or "traffic jam" just by the co-occurrence of objects in a scene (Daniels and Metaxas, 2018). Also the help of depth information in an image can improve the overall understanding of the scene (Zhang et al., 2017). From here we can now go into even more detail by creating scene graphs, explicitly stating relationships between objects. These will be described in section 1.1.3.

FIGURE 1.2: *(Top left)* Object detection. *(Top right)* Scene segmentation performed on the image segmenting people from the rest of an image. *(Bottom left)* Object recognition. *(Bottom right)* Scene understanding.

1.2.2 Scene Graph Generation

Scene graph generation aims at extracting semantic information from an image which states the relationship between the objects detected in the image. A scene graph G = (O, E) contains a set of objects O = {o_0, ..., o_n} and a set of edges E $\subseteq$ O$\times$ R $\times$ O. R is a set of relationships {r_0, ..., r_n}. An object has the form o_i = (c_i,A_i) in which c_i represents the object's class and A_i represents the object's attributes. A scene graph can be grounded in an image, meaning every object in the graph corresponds to a specific area in the image, i.e. a bounding box (Karpathy, 2016). It is possible to train a neural network to create a scene graph grounded in a given image (Xu et al., 2017). These scene graphs can then be used for image retrieval via specific queries (Karpathy, 2016) or to create a knowledge base to make visual query answering possible (Zhu et al., 2015).

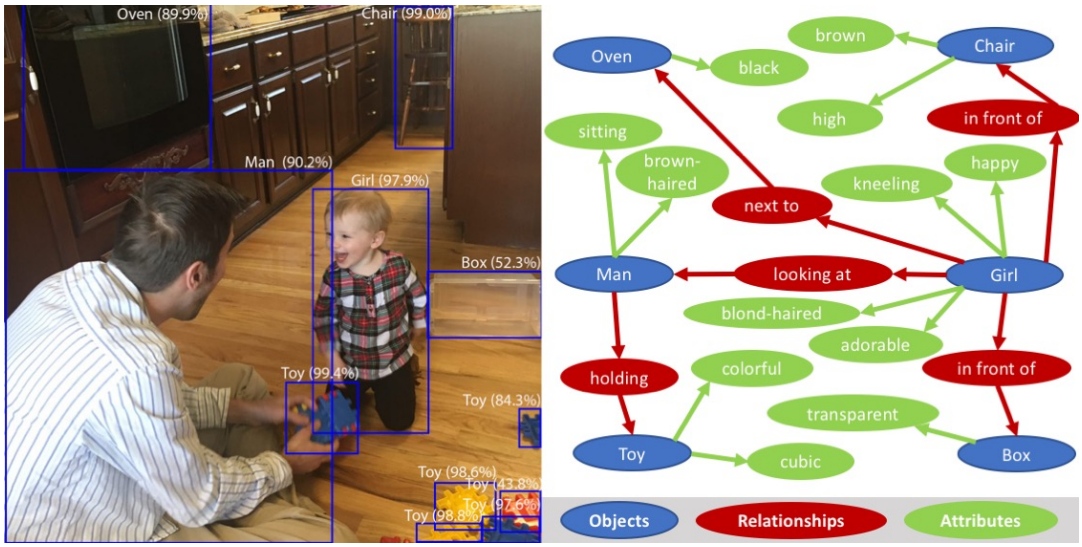

FIGURE 1.3: Example for a scene graph. *(Left)* Image with bounding boxes and object labels. *(Right)* Part of corresponding scene graph. Blue: Objects, Red: Relationships between two objects, Green: Attributes of an object.

In figure 1.3 there is an example of a scene graph with objects (blue), relationships (red) and attributes (green). The grounding of this graph would be all references from the objects to the corresponding bounding boxes in the image on the left.

1.2.3 Progressive Learning and Catastrophic Forgetting

One of the major problems in implementing a real-world scene segmentation system is to let it learn new classes over time. This system needs to be flexible enough to incorporate a newly learned class while not decreasing its performance on the already learned classes. When using neural networks this would imply a network which can add more and more nodes to its output layer but still keeps the same activation patterns for the neurons which are already present. This is a very hard task and previously unsolved on such a large scale. There has been a previous attempt to implement a progressively growing neural network (Venkatesan and Er, 2016) but it only deals with a maximum of 10 classes, while a real-world object recognition task would involve several thousand classes. Additionally, this implementation uses an extreme learning machine which would be hard to combine with the rest of the planned model architecture.

Overall the task of learning new classes on the fly is important for real world applications, yet still remains relatively unexplored. The problem of catastrophic forgetting results from the stability-plasticity dilemma (Abraham and Robins, 2005). Meaning that the network weights need to be sufficiently stable to avoid forgetting previously obtained knowledge while remaining flexible enough to learn new tasks.

There have been several methods introduced to combat this problem. One idea is the regularization in which some weights are manipulated more than others (Hinton and Plaut, 1987). The idea is to take plasticity away from nodes that contribute most to the learned task and only adapt weights which do not contribute much (Kirkpatrick et al., 2017). Another method is to train an ensemble of networks on different tasks and then combine those by using majority voting Polikar et al., 2001. This however, naturally increases the memory requirements with each additional task that is included.

Another approach, which is similar to the approach I will be taking here, is to rehearse the old tasks while training the new one (Ratcliff, 1990). This approach can be augmented by using the network to compile pseudo-patterns so that storing all past data can be avoided (Robins, 1995). Rehearsal can also be combined with a dual memory approach that is derived from the human brain which stores newly formed memories in the hippocampus and then gradually transfers them into the pre-frontal cortex. Some of these models even incorporate a sleep phase into their training to accomplish this transfer (Gepperth and Karaoguz, 2016). At last, several sparse coding methods have been tried out (Murre, Phaf, and Wolters, 1992, Krushke, 1992, Coop, Mishtal, and Arel, 2013). Albeit all of these ideas, (Kemker et al., 2017) claims that the problem of catastrophic forgetting is still unresolved because many of these approaches don't scale up to bigger data sets or have other significant short comings.

2 Methods

All experiments performed are related to image recognition and essentially measure how well a network learns to classify an image based on its pixel values. As a measurement for this, the percentage of correct classifications is used. Throughout all modifications of the experiments, I keep as many variables constant as possible to retain good interpretability of the results. I will now go through the basic set up.

2.1 The Network

All experiments are performed on the same network with only a few modifications made. The network always consists of two convolutional layers followed by two fully connected layers, while the last fully connected layer is the read out layer matching the one-hot encoded [1] target class of an image.

The first convolutional layer contains 16 kernels with the shape 5x5x3 (or 5x5x1 for black and white stimuli). The second convolutional layer is made of 32 3x3x16 kernels. Both convolutional layers are followed by a pooling layer using 2x2 maximum pooling which is an efficient way to achieve more location invariance in object detection (Scherer, Müller, and Behnke, 2010).

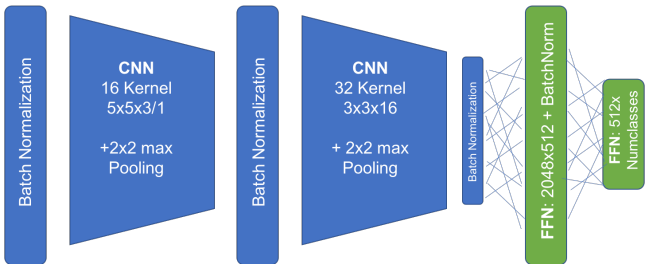

FIGURE 2.1: Network structure. (Blue) CNN part of the network. (Green) Fully connected part of the network.

The second convolutional block is then flattened to shape 2048x1 and followed by a fully connected layer of shape 2048x512. The resulting 512 activations are then multiplied with the weights of the last fully connected layer to obtain the final activations of the network. This layer shape depends on the N number of classes in the data set (512xN). These N final activations are then processed by a SoftMax activation function to generate a probability distribution over all classes and pick the class with the highest probability.

Before each major processing block in this network, batch normalization is performed for faster training, more stability and as a slight regularization (Ioffe and Szegedy, 2015). To do this, you subtract the mean of the output of a layer from the output itself and divide it by its standard deviation before feeding it into the next layer. For the activation function in each layer rectified linear units (ReLUs) are used which achieve quality results while having a higher biological plausibility than for example tanh activations (Glorot, Bordes, and Bengio, 2011). The network

[1] A one-hot label consists of N zeros where N is the number of possible classes-1 and a one at the position i in the one-hot array which represents the numeric label. So if the numeric label is 3 and there are 5 possible classes the one-hot vector v has shape (5) with 4 zeros and a one at v[3] such that v=[0,0,0,1,0].

is trained using an Adam optimizer with a learning rate of 0.0001. This optimizer was chosen because it achieves good and fast results on a wide range of non-convex optimization problems without requiring much parameter fine-tuning (Kingma and Ba, 2014). I've also tried other optimization techniques such as standard gradient descent (SGD), SGD with momentum and AdaGrad, but none of them could achieve comparable results in the same amount of training steps (figure A.1). The optimizer aims at minimizing the cross-entropy between the network activations and the target activations while the target activations are a one-hot encoding of the desired class label.

The network is always trained with a mini batch size of 200 images. One epoch is finished after each image of the training set was shown once. The order of the images in each epoch is random and no image is shown twice in one epoch. One mini batch is not necessarily balanced between classes but overall each class shows up equally often in one epoch. Every 50 steps (50 mini batches of the training data), a validation is performed during which all images of the validation set are shown in one big batch and the average performance on the complete set is calculated. The network is not trained on the validation set and therefor no weight updates are performed. After training of the network is completed the whole test data set is shown to the network once to calculate an overall test accuracy which is then used to quantify the overall performance of the network under certain conditions and compare between condition.

2.2 The Data Sets

To increase the likelihood of universality in the findings made, experiments are performed on three different data sets. All data sets imply a task of image recognition and each sample maps one image to one of n>=10 classes. CIFAR-10 and CIFAR-100 can be downloaded here. The MNIST data set can be downloaded here.

2.2.1 CIFAR-10

CIFAR-10 (as well as CIFAR-100) is a subset of labeled, colored images collected from the tiny image data set collected by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Each image has the format 32x32x3 and one of ten distinct class labels assigned to it. Figure 2.2 shows a few random stimuli of the CIFAR-10 data set. Overall, there are 60.000 images in the data set, 6.000 for each class. 50.000 of these belong to the training set and 10.000 to the test data set. Both data sets are balanced between classes. To track overfitting during training, a subset of the training data was randomly selected as the validation set. This makes up a tenth of the training data, adding up to 5000 images. It is chosen once, and the stimuli selected for the validation set are never shown during training.

FIGURE 2.2: Sample stimuli of the CIFAR-10 data set with its corresponding labels.

2.2.2 CIFAR-100

CIFAR-100 is very similar to CIFAR-10. It also contains 60.000 32x32x3 images with one class label assigned to each of them. In contrast to CIFAR-10, there are 100 distinct class labels instead of ten which means that there are only 600 images for each class in the data set which is a tenth of CIFAR-10. Due to the split into training and test set there are just 500 images per class to be used during training (50 of which are designated to validation and never used for the network to train on) and 100 per class for testing. The smaller amount of data leads to more overfitting which means that certain countermeasures need to be taken.

2.2.2.1 Data Augmentation and Dropout

In this case, data augmentation is used. Data augmentation is a popular technique to increase the amount of data available for training and to decrease overfitting. I apply random rotation, random crop, crop and padding, horizontal flip, Gaussian blur, sharpening, coarse dropout and the addition of small random values. These modifications are applied randomly to 70% of the images during training and validation. Each time an image is picked for modification, two out of the eight possible modifications are picked and applied. During random rotation, the image can be rotated slightly by -45-45 degrees. In cropping the image is randomly cropped from each side by 0-16 pixels. Crop and padding randomly selects pads on the image covering 25% of it and replaces them with zeros. Horizontal flip mirrors the image on the y-axis. Gaussian blur is applied with the random value sigma which can be between zero and one. Sharpen runs a sharpening kernel over the image with a lightness between 0.75 and 2 and mixes this with the original image with an alpha value between zero and one. Coarse dropout randomly replaces 2-35% of the pixel values with zeros. Finally, the addition of small random values adds random values between -40 and 40 to each pixel. Figure 2.3 shows the original CIFAR-100 stimuli (left) after applying data augmentation (right).

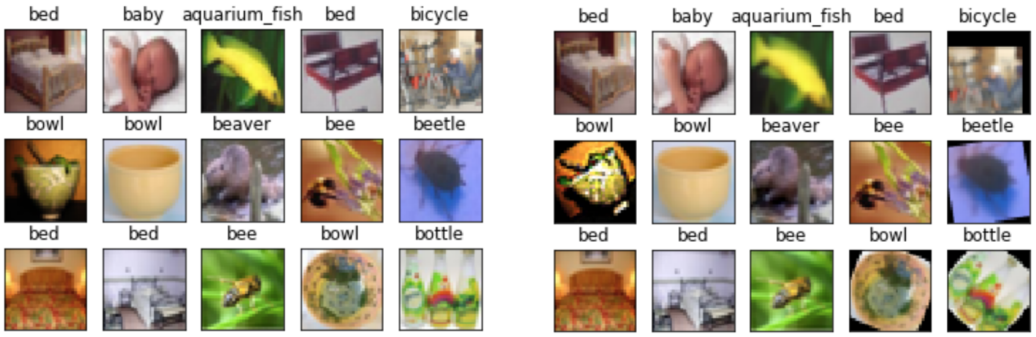

FIGURE 2.3: (Left) Sample stimuli of the CIFAR-100 data set with its corresponding labels. (Right) Sample stimuli of the CIFAR-100 data set with its corresponding labels after data augmentation.

In addition to the data augmentation dropout is introduced into the network and applied randomly to 30% of the neurons in the last layer during training and validation. Both of these measures combined lead to a significant reduction of overfitting and an increased accuracy on the test set (figure 2.4). When measuring the test accuracy, no data augmentation or dropout is used which is why it is overall better than the validation accuracy at the end of the network training.

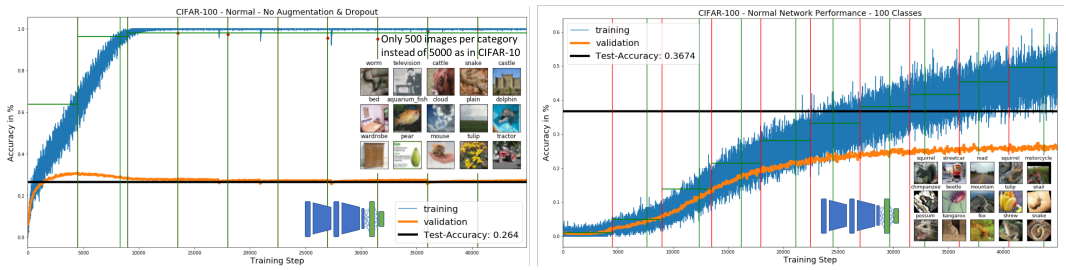

FIGURE 2.4: (Left) Network performance on CIFAR-100. (Right) Network performance on CIFAR-100 when using data augmentation and dropout as explained above. Blue: Training performance, Orange: Validation performance, Black: Test performance at end of training. (Vertical lines will be explained in section 2.5.)

2.2.3 MNIST

The MNIST dataset consists of 70.000 images (60.000 training, 10.000 test) of handwritten digits from zero to nine. The images 28x28 pixels big and don't contain color which reduces the third dimension from three to one. This data set is much easier for the network to learn and after a few epochs it reaches an accuracy close to 100%.

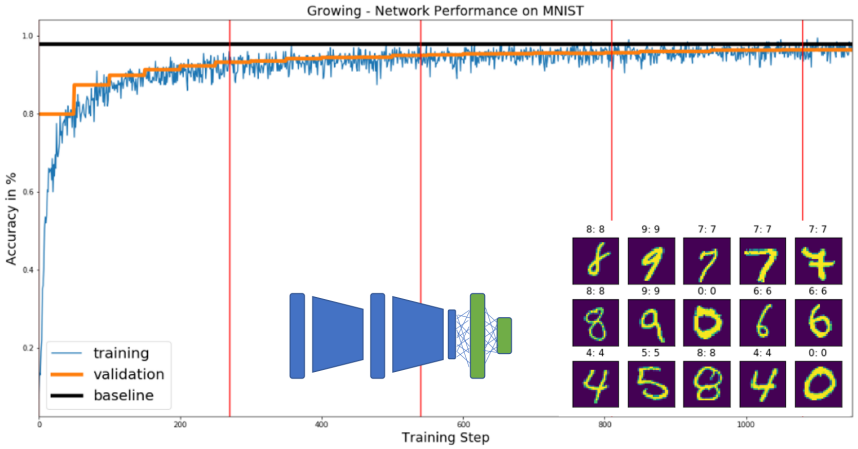

FIGURE 2.5: Performance of the network on the MNIST data set. Blue: Training performance, Orange: Validation performance, Black: Test performance at end of training. Vertical Line: Start of a new epoch

2.3 Iterative Class Introduction

Sometimes not all the data is available during the beginning of training in real world applications of deep neural networks. It is possible for new classes to show up while a network is used and these classes should be incorporated into the network without having to retrain everything from the beginning again. The goal is to have a network which can keep learning new classes and grow with them without forgetting the classes it already knows.

To investigate continuous learning, the first test condition works with a static network. The network does not change size during training and starts out with an output layer of the size M where M is the maximum number of classes that will be in the data set. At the start of the training, only a subset of the classes will be shown and some of the output nodes remain unused. During the course of the training new classes will get introduced until all classes are represented in the training data and all nodes of the output layer are used.

2.4 Growing a Network

The continuous learning described above can fulfill the criterion that new class introduction at a later point in training is possible. However, in a real-world scenario, one usually does not know how many classes will be present in the end and would have to start training with an output layer that is much too big for the number of classes in the training set. This would be redundant and consume more resources than actually needed.

To solve this problem, a new neuron could be introduced into the last layer with each new class added to the data set. This will be the second test condition for continuous learning. Since TensorFlow doesn't allow changing the size of a layer during training, a little workaround was used in which the last layer was reinitialized with more nodes whenever it is needed.

2.5 Plotting Conventions

For consistent comparisons I plot all the experiments in the same way. Since it is necessary to visualize very complex information, the three basic types of plots used will be introduced in this section. They will consistently have the same meaning throughout the whole thesis. The data used to create the plots in this section will be explained in more detail in the results section. For now, they just serve the purpose to explain everything about all components visualized in the plots.

2.5.1 Performance Plots

The performance plots contain three panels. The top panel displays the performance of a network under specific conditions over time. On the abscissa we have the training steps and on the ordinate, the accuracy of the network. One training step represents the processing of one mini batch (200 images). The accuracy is calculated by dividing the number of correct predictions by the mini batch size and thereby lies between 0 and 1. The blue line represents the performance of the network on the training data while the orange line shows the performance of the network on the validation data. The individual validation measurements are connected by straight lines since validations are only performed every 50 steps for better visibility. However, the validation accuracy between two measurements in this plot does not have to lie on this connecting line. Finally, the straight black line represents the accuracy of the network on the test data set at the end of the training. The exact value of it is given in the legend.

Sometimes the top panel also contains these blue squared and orange circles. These are intended to show which condition is shown in the plot and what exactly is happening in the network during training. The blue boxes represent the last output nodes of the network. Each circle depicts one node and the number of circles in the blue boxes represent the number of nodes in the last layer of the network. In this example, we see that the layer size grows over time. In the beginning the last layer contains six nodes and in the end, it contains ten while introducing only one new node at a time. The orange circles tell how many classes are represented in the data set. In this case, there are always as many classes in the data set as nodes in the network. In the first continuous learning condition the number of circles/nodes in the layer will stay the same (ten nodes from beginning until the end) while the number in the orange circle will increase since new classes are being introduced.

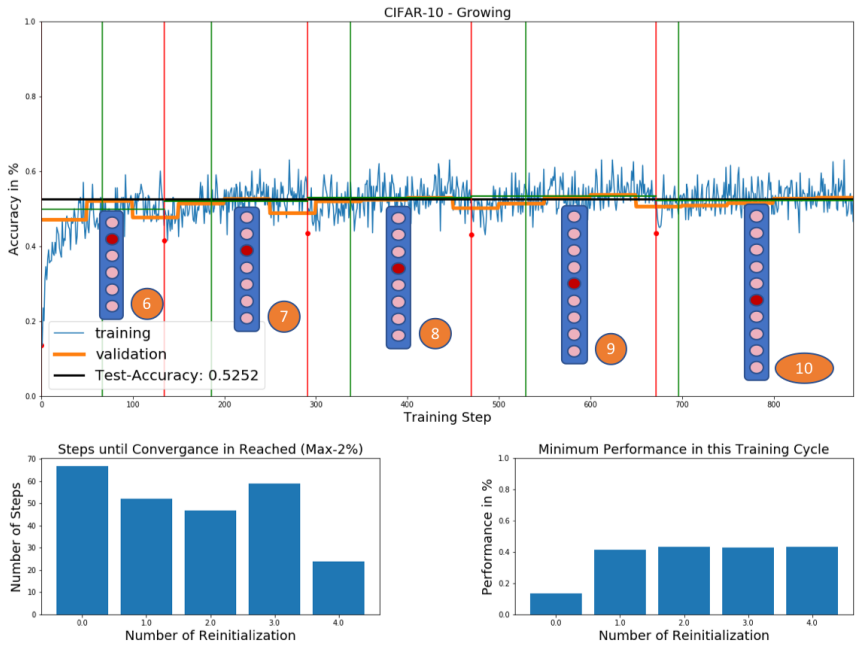

FIGURE 2.6: Example plot to explain the plotting conventions. This plot shows a growing network trained on CIFAR-10

The red lines in the plot represent the start of a new epoch or a new training cycle. This is when a new class and/or a new node can be introduced. In CIFAR-10, one of these training cycle ends after each stimulus has been shown once. In CIFAR-100 one of these training cycles ends after each stimulus has been shown 20 times. The red dot marks the training performance at the start of an epoch. This value is visualized in the right panel on the bottom. When discarding outliers, the performance at the beginning of an epoch is also the lowest performance in this training cycle. When looking at the bar plot on the bottom right we can thus see how low the performance is at the start of an epoch after for example, a new class has been introduced. Each bar represents the minimum performance in one of the epochs, with the first epoch being the leftmost bar and the last epoch the rightmost bar.

At last, a measure of convergence is defined and plotted in the bottom left panel to determine how long it takes the network to recover from a new class introduction. Convergence is defined as reached when the network reaches a performance of max(Performance in this epoch)-2%. To avoid outliers from influencing this measure, the function is first smoothed with a moving Gaussian of 20 degree. Figure 2.7 shows some sample training accuracies and the corresponding smoothed function.

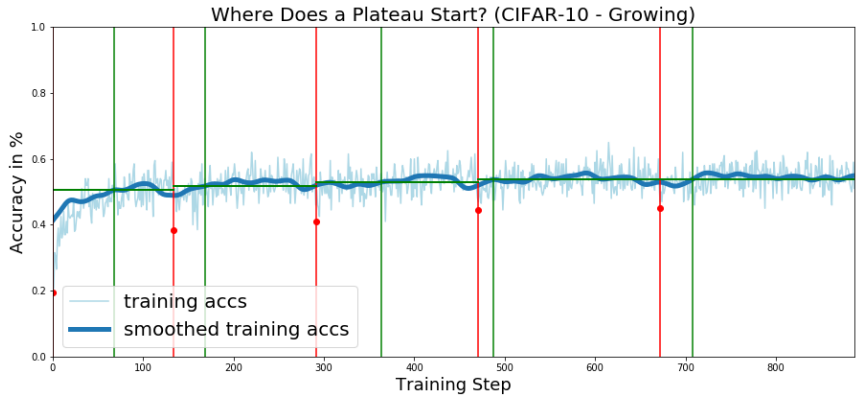

FIGURE 2.7: Where does a plateau start? Explained on a growing experiment performed on CIFAR-10. The CIFAR-10 original training accuracy's are shown in light blue. The accuracies smoothed with a 20-degree moving Gaussian are shown in dark blue. Corresponding plateau starts marked with green lines. Horizontal green line: EpochMax-2%. Vertical green line: Step at which EpochMax-2% is reached on the smoothed accuracy function.

In the accuracy plots (figure 2.6) the step at which the epoch maximum -2% is reached is marked with a vertical green line. The epoch maximum -2% of the respective epoch is marked with a horizontal green line. The positions of the vertical green lines are then visualized in the bar plot on the bottom left with one bar per epoch.

2.5.2 Old-New Class Comparison

To investigate if a new class introduction influences the accuracy on the old classes, they are plotted separately from the accuracy on the new class. In figure 2.8 you can see such a plot. In this plot, the accuracies have been smoothed with a 5 degree moving Gaussian [2] for better visibility since especially the accuracy on the new class has a big variance. In some of the following plots there is no smoothing applied to the old-new-accuracy curves and if there is it is explicitly mentioned.

FIGURE 2.8: Example plot for an accuracy comparison between old and new classes.

[2]Method from https://www.swharden.com/wp/2008-11-17-linear-data-smoothing-in-python/ used.

In all plots of this style you can find three accuracy curves. A blue curve is for the accuracy on the old classes, an orange curve for the accuracy on the newly introduced class(es) and a green curve with the accuracy's over all classes. The black line represents the test accuracy again and the red lines visualize the start of a new training cycle.

2.5.3 Repeated Runs

Due to random initializations of the weights and random order of the presentation of images, the network doesn't always reach the same test accuracy at the end of the training. When using the same order of class introduction, the test accuracies can have a difference of up to 5% just due to the random factors named above. To account for this variance, I run all experiments on CIFAR-10 a hundred times under the same conditions. The distribution of the hundred test accuracies is then compared to the accuracy distribution of a hundred runs under a different condition. Figure 2.9 shows the plot of such a 100x experiment repetition.

FIGURE 2.9: Example plot for a plot showing 100 repeated runs on CIFAR-10.

The colored curves each represent the training accuracies of one of the hundred runs. The black horizontal lines visualize the hundred test accuracies obtained from each of these runs. In the legend, the average test accuracy is named as well as the smallest and the largest test accuracy. Sometimes the same blue and orange helper symbols as explained in section 2.5 are added for an easier understanding of the conditions under which these results were obtained.

2.6 Freezing Network Layer

In some of the experiments, there will be discussion concerning the freezing of some of the network layers. This means that at a certain point some of the layers stop getting their weights updated so they stop learning. This is interesting to see if after a certain time a layer has learned enough to provide sufficient information to the next layers for them to make good decisions or even learn new classes. In the experiments, the convolutional layers are frozen which is intended for feature extraction. If these layers stop getting weight updates they keep extracting the same features they learned on the first classes. If they learned to extract universal features then this should not be a concern for the overall network performance since it should

still provide the fully connected layer with useful features for image classification. Although if the learned features are very specific to the first classes, they do not assist with the newly introduced classes and thereby lead to a lower accuracy on those classes.

Freezing layers is rather simple with TensorFlow. One can simply assign the optimizer a list of variables that it should optimize during training. If the variables are removed from this list, they will stop being optimized and keep the weights that they already have.

3 Results

Due to the very high performance on the MNIST dataset I will use CIFAR-10 and CIFAR-100 for most of my experiments because with those datasets the performance has more room to improve and the differences between conditions can clearly be examined. Nevertheless, I will also demonstrate some of the general effects on the MNIST data set to show that they do not only apply to the CIFAR data sets. The CIFAR-10 data set will be used for all general experiments that can be performed on a data set with ten classes since training on this data set is relatively fast compared to CIFAR-100 and can be performed multiple times. CIFAR-100 will mainly be used to demonstrate how the effects on CIFAR-10 translate to a data set with more classes and what effect the introduction of multiple classes at once can have.

3.1 Weight Initialization for a Growing Network

One property of a growing network is that during the course of training new neurons and weights connecting to them get introduced. Additionally, in the procedure used, the old nodes in the same layer get reinitialized due to some static limitations of TensorFlow. These newly initialized weights of the network need their initial values to be assigned. To figure out how to best initialize them, they are compared in three conditions.

Random initialization (Gaussian normal distribution with variance of $\sqrt{512}$ where the number 512 originates from the amount of neurons connecting into the layer also called *fan-in*) is compared with 0.01 initialization and old weight initialization. For the old weight initialization, the previously trained weights from the last step are saved and assigned to the new layers weights. This means that all the connections which go to the neurons presented in the last layer before re-initialization stay exactly the same. The connections to the newly added neurons are initialized with weights randomly sampled from the old weights. A comparison of these three weight initializations over five epochs of training on CIFAR-10 can be seen in figure 3.1.

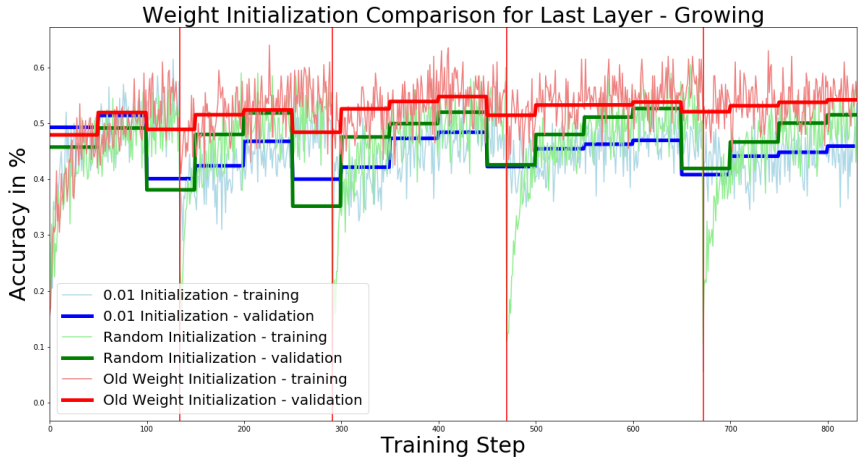

FIGURE 3.1: Comparison of network performances on CIFAR-10 with different weight initializations. These networks are trained with continuous class introductions and growing of the last layer. The weight initialization determines how the edges connecting to the nodes of the last reinitialized layers start out at the beginning of an epoch.

As seen in figure 3.1 old weight initialization seems to be the most effective approach. It makes sense because random or 0.01 initialization discards all the previously learned weights and the last layer basically has to start from scratch again. It recovers quickly since the other layers stay the same and only the last layer needs to be retrained. Although when keeping the weights from the last layer, the drop is significantly smaller.

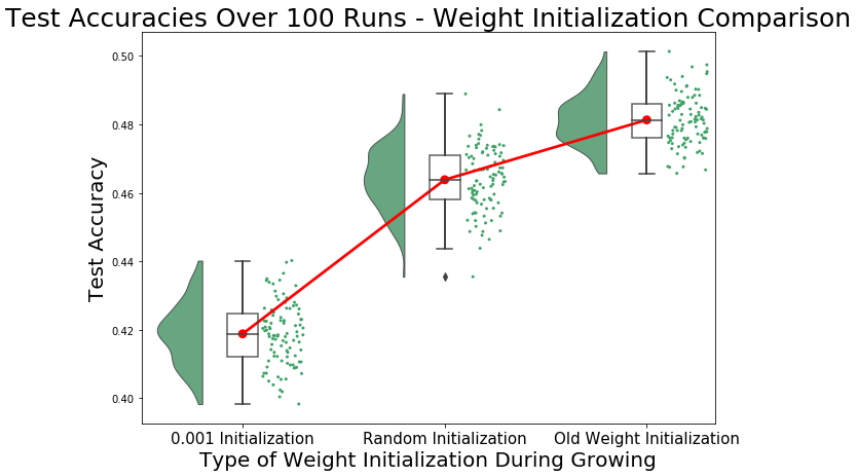

FIGURE 3.2: Comparison of network performances on CIFAR-10 with different weight initializations over 100 runs. These networks are trained with continuous class introduction and growing of the last layer.

Running the same three experiment conditions a hundred times produces similar results (figure 3.2). The old weight initialization produces the highest test accuracies after five epochs of network growing. Initializing the weights randomly is better than initializing all weights with a fixed constant (here 0.001). The old weight initialization ($M=0.48$, $SD=0.01$) is significantly better than the two other initialization techniques, $p<0.001$. Therefore this type of weight initialization will continue to be used for all following experiments.

3.2 First Comparison of the Three Conditions on CIFAR-10

As mentioned in section 2.3 I will be testing the performance of the network under three main conditions. Two of those involve continuous learning. The first condition, labeled as *Normal*, can be seen as the baseline condition which shows how the network performs on the complete data set without any new class introductions. The network therefore trains on the full data set starting from the beginning onward. In the second condition, here labeled as *No Growing*, the network starts training on a subset of the classes and with each training cycle, a new class is introduced. The network itself however never changes. The output layer always has the same size, with one output node for each possible class in the end of training. In the CIFAR-10 example this means that it always has ten output nodes even though in the beginning only the first six classes are contained in the training data. In the third condition, called *Growing*, the last layer of the network changes size according to how many classes are contained in the data set. So, in the beginning of training we have six classes in the data set and six output nodes in the last layer. When a new class gets introduced into the data set the output layer also grows one node bigger.

The following figures show one run under each of these three conditions. In figure 3.3 you can see the normal performance of the network on the whole CIFAR-10 data set over five epochs. This means at the end of training each stimulus of the training data has been shown five times. A continuous increase occurs in the accuracy of the network which flattens towards the end of training. The trained network reaches a test accuracy of 0.55.

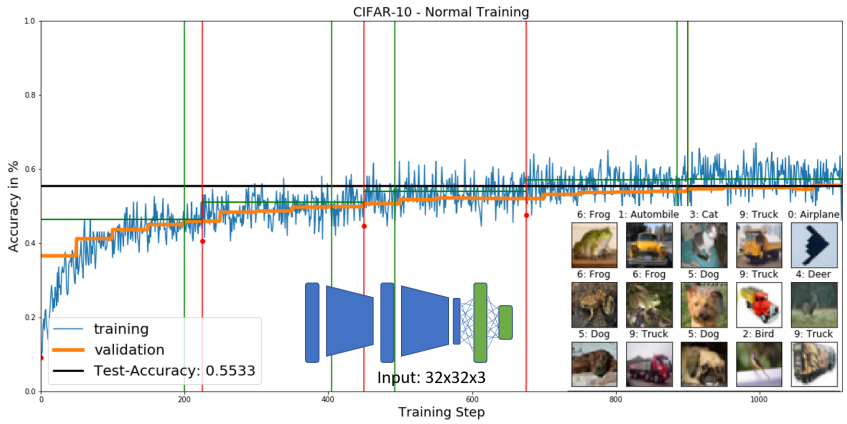

FIGURE 3.3: Network performance on CIFAR-10. Training on all ten classes from beginning on over five epochs.

Figure 3.4 shows the performance of the same network with consecutive class introduction. In the first epoch the training data only contains classes 0-5. In the second epoch, class six gets added to the training data and so on until in the fifth epoch all ten classes are present in the data set.

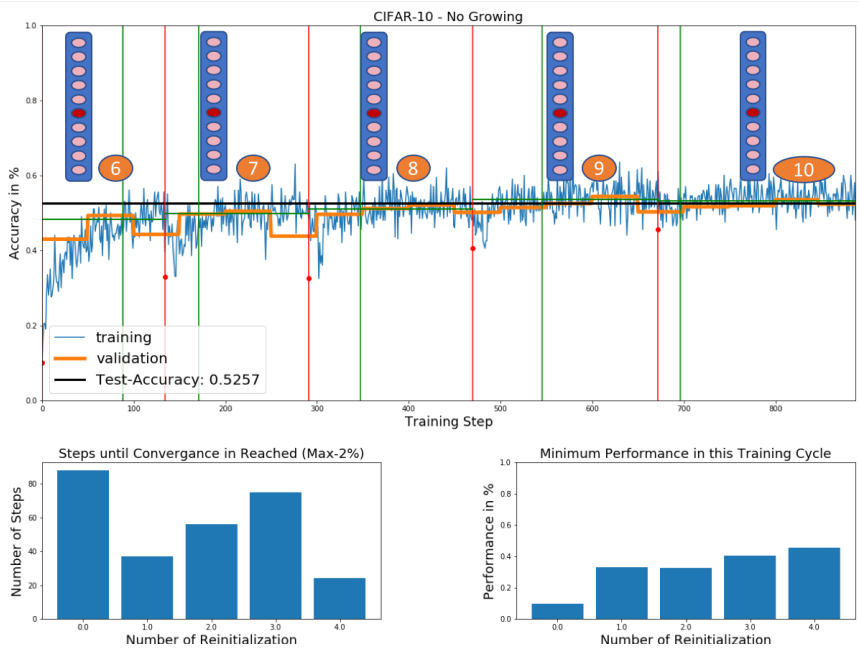

FIGURE 3.4: Network performance on CIFAR-10 when iteratively introducing new classes. Starting out with six classes and adding one new class each epoch until all ten classes are in the data set. The output layer always consists of ten activations.

We can see that there is a small dip in performance after each new class introduction (red lines) from which the network then quickly recovers. Converging performance is still always reached in the first half of each epoch and after initial training in the first epoch the performance does not drop below 0.3 anymore which is clearly above chance. At the end of training the network reaches an overall test performance of 0.53.

Figure 3.5 shows the performance of the network with the same class introduction structure as figure 3.4. Here the difference is that the number of nodes in the last layer change with the number of classes in the data. So, when we have six classes in the data set of the first epoch, the output layer also only contains six output nodes. This reduces the redundancy of unused nodes and provides the ability to show that after training, new classes can be added even when it was not already expected and built into the network structure.

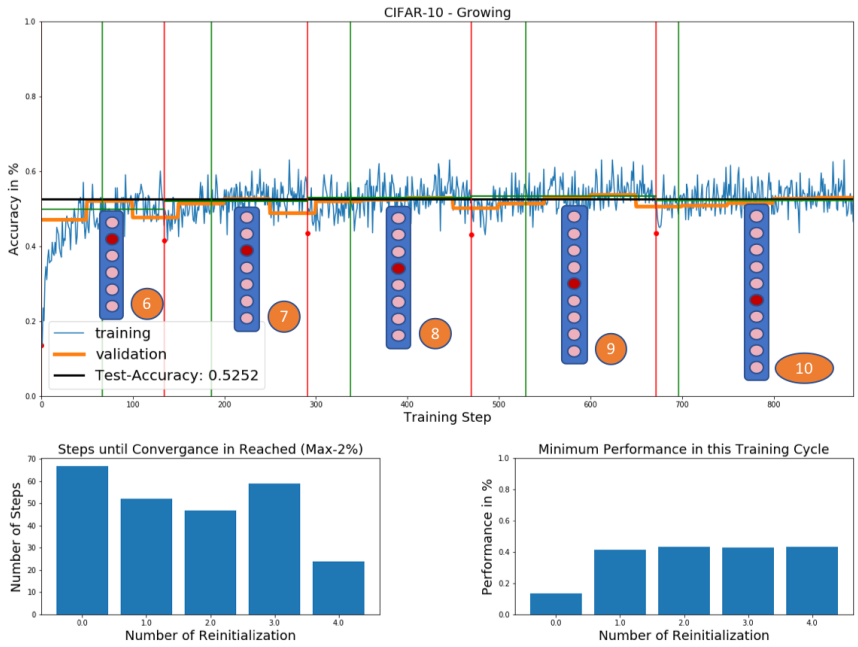

FIGURE 3.5: Network performance on CIFAR-10 when iteratively introducing new classes and growing the last layer. Starting out with six classes and six output nodes, adding one new class and output node each epoch until all ten classes are in the data set.

The results look very similar to the figure above where the number of output nodes stayed constant. We also have a little dip in performance at the beginning of each epoch from which the network then quickly recovers. The overall test performance is also very similar with 0.53.

Overall, we can see that a new class introduction does not lead to a reset of the network. The performance always stays above chance after the initial training in the first epoch. The dip in performance seems to only come from the challenge of learning a new class. I will investigate this further in the next section (3.2.1).

The test performances show no difference between a growing and a non-growing last layer. However, the normal training without new class introductions seems to perform better than a network with class introductions. To quantify these findings, the same tests will be ran a hundred times in section 3.2.2.

3.2.1 Performance on Old vs. New Classes

To investigate where exactly the performance drop after a new class introduction comes from, the accuracy will be plotted on the old classes and new class separately. Figure 3.6 shows such a plot for the accuracies of a growing network. The comparison of old and new accuracies for a non-growing network with continuous learning looks very similar and can be found in the appendix (A.4). All accuracy curves have been smoothed with a five-degree moving Gaussian for better visibility. A non-smoothed version can also be found in the appendix (A.5).

FIGURE 3.6: Performance comparison between old and new classes on CIFAR-10. One new class is introduced each epoch (red line) and the last network layer grows with the number of classes in the training data. Accuracies are smoothed.

You can see that the performance on each of the new classes starts out low at the beginning of the epoch and then increases up to the performance of the old classes. The performance on the old classes stays relatively unchanged. This means that the dip in overall performance after a new class introduction is caused mainly by the network having to learn the new class and not by damage done to the already learned classes. The same effect could be shown on the MNIST data set (figure 3.7).

FIGURE 3.7: Performance comparison between old and new classes on MNIST. One new class is introduced each epoch (red line) and the last network layer grows with the number of classes in the training data. Accuracies are not smoothed.

3.2.2 Multirun Comparison and Statistical Significance

Since we always have some variance in the test accuracies depending on random network initializations and the random mini batch composition during training it makes sense to repeat the experiment multiple times to obtain a stable average for each of the conditions. In this case, each of the three conditions are ran one hundred times. The obtained plots can be seen in figure 3.8.

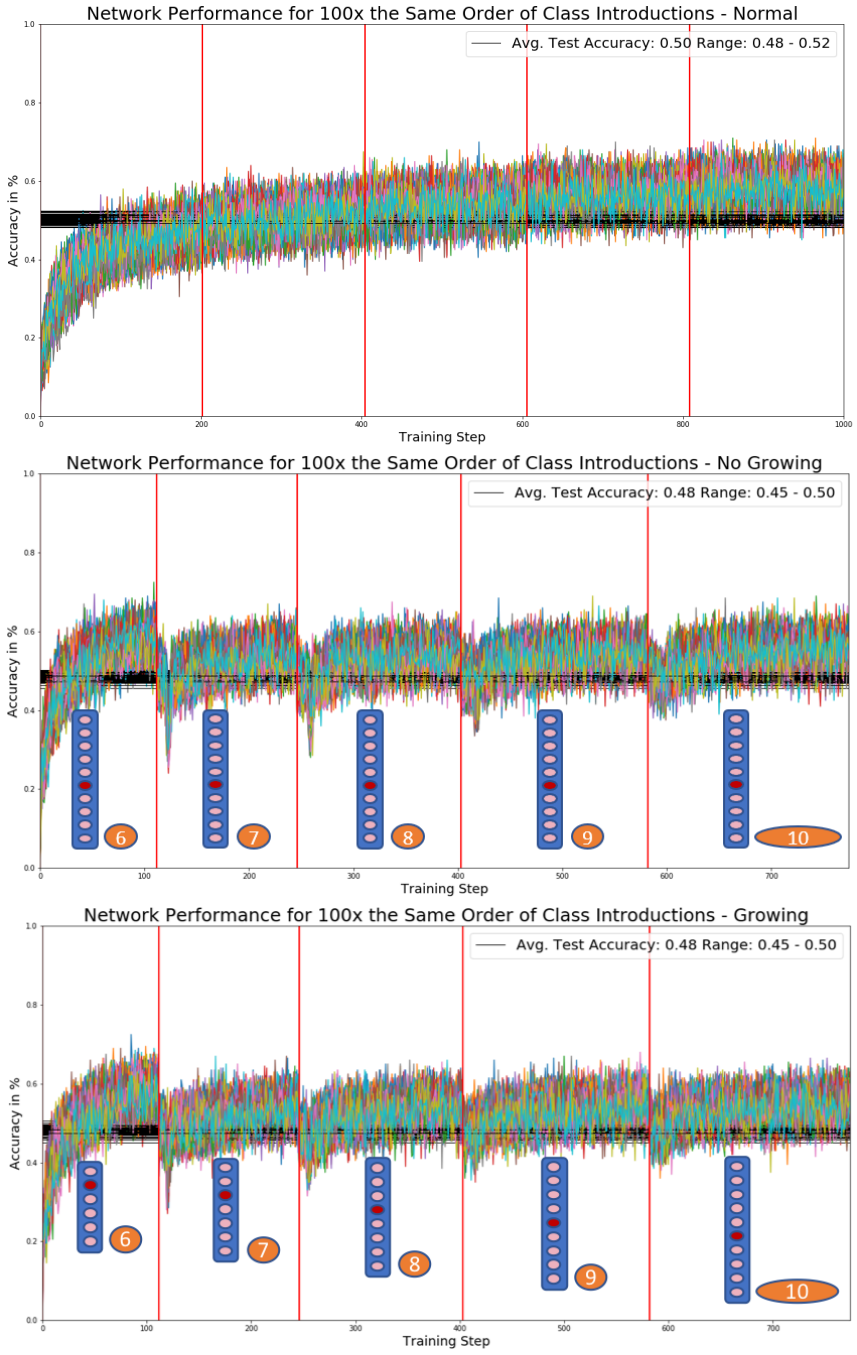

FIGURE 3.8: 100 runs under each of the three conditions. (Top) Normal training - 10 classes and output nosed from beginning on. (Middle) No growing - Starting with six classes in the training set up to ten in the end. Output layer has always ten nodes. (Bottom) Growing - Number of classes and number of nodes in the output layer increase from six to ten over the course of training.

You can see a variance of 4-5% in the accuracies caused by the random initializations. The average test accuracy for the continuous learning conditions are the same when cut off at two decimal points (0.48). The test accuracy for normal training of 0.5 is 0.02 higher than for the two continuous learning conditions. In figure 3.9 a comparison is presented between the test accuracy distributions for the three conditions.

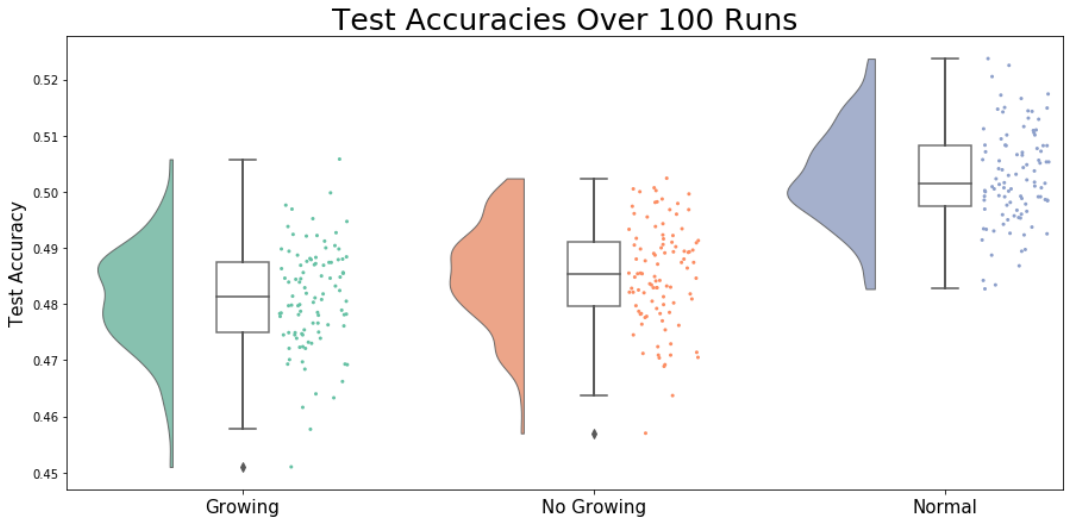

FIGURE 3.9: Performance comparison between all three conditions. Data distributions are obtained from 100 runs under each of the three conditions.

Looking at the two distributions shows a visual difference between the normal training and the two continuously learning conditions (growing and non-growing). A one way analysis of variance shows that the accuracies are significantly related to the condition, $F(2,297)=163.59$, $p<0.001$. Post hoc t-tests between the three conditions show significant differences between all conditions where both comparisons to the normal training ($M=0.50$, $SD=0.01$) are highly significant with $p<0.001$. The difference between non-growing ($M=0.49$, $SD=0.01$) and growing ($M=0.48$, $SD=0.01$) is also significant, $t(99)=3.096$, $p=.003$. This shows that the normal network training reaches a significantly higher performance than the continuously learning conditions after five epochs of training. The condition in which the last layer grows in size has the lowest performance and is on average lower than the performance on a network where all neurons in the last layer are already initialized.

3.3 Network Growing on CIFAR-100

The CIFAR-100 data set offers the opportunity to experiment with different amounts of new classes that can be introduced and makes it possible to show if the previous findings can generalize to bigger data sets. Since the training on CIFAR-100 takes several hours all results presented here are just individual runs without accounting for variance between runs and performing significance tests. Nevertheless, the findings can show that the basic principles found on CIFAR-10 can be applied to a bigger data set such as CIFAR-100.

As previously mentioned in section 2.2.2 the CIFAR-100 data set has ten times less examples per class than the CIFAR-10 data set such that the network is trained using data augmentation and drop out. Due to the higher complexity of the task each epoch consists of 20 runs through the data set. This means that after one epoch each image from the training data has been shown 20 times. I also introduce an initial phase of pretraining when working with more than ten classes that consists of several epochs (usually five) in which no class introductions are happening so that the network can reach an initially good performance on the classes it starts out with before new classes are introduced.

3.3.1 Introducing Different Number of New Classes

The results shown in this section are all displaying the network performance of a growing network on the CIFAR-100 data set with variations in the number of classes the network starts out with and ends with. Since there is no variation possible for the normal condition just refer back to figure 2.3 for the network performance when training with 100 classes all the way through. The results for the continuous learning condition without growth can be found in the appendix (A.3.1) but, just as for CIFAR-10, they have a similar resemblance to the results of a growing network.

First, an examination of the network performance will be conducted under the same conditions as in the previous results on CIFAR-10. This means to start with six classes and add one class each epoch until ten classes are in the training data.

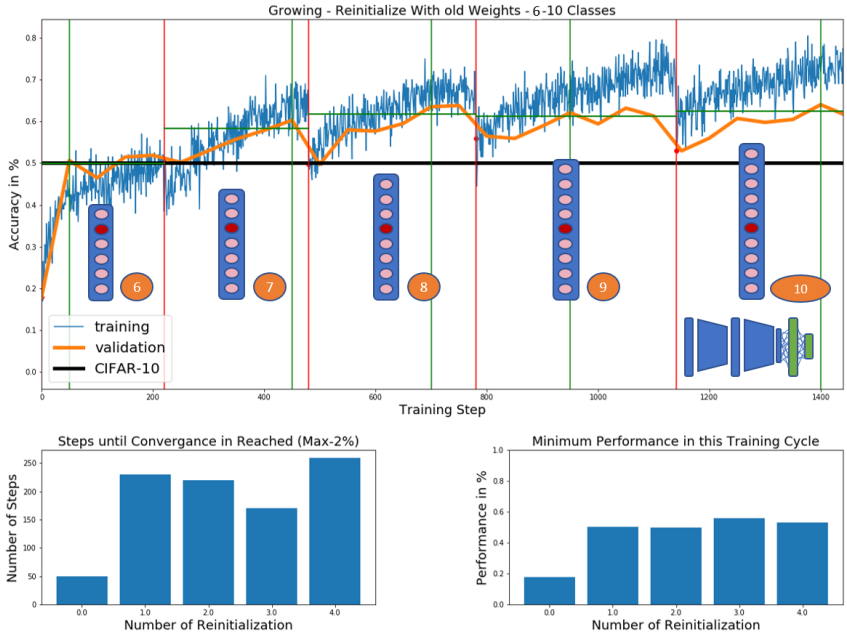

FIGURE 3.10: Training accuracies on CIFAR-100. Starting training with six classes and introducing one class each epoch until ten classes are in the training data. The network is continuously growing with each new class introduction.

Similar behavior of the network can be seen as observed with the CIFAR-10 data set. When a new class is introduced, the performance drops shortly but then quickly recovers. It never drops below chance after the first epoch of training. The difference in the CIFAR-100 data set is that much more overfitting can be seen here. The training performance is way higher than the validation and test performance. This is a result of the relatively smaller size of the data set compared to the number of classes it contains. However, since the main results (small performance drop, fast recovery) are the same the overfitting should not be a big problem for the future results.

Now an examination of how the same procedure would look if more classes are present. Figure 3.11 shows the performance of a growing network which has been pretrained on 95 classes for five epochs. Starting from the sixth epoch one new class gets introduced until all 100 classes are in the training set.

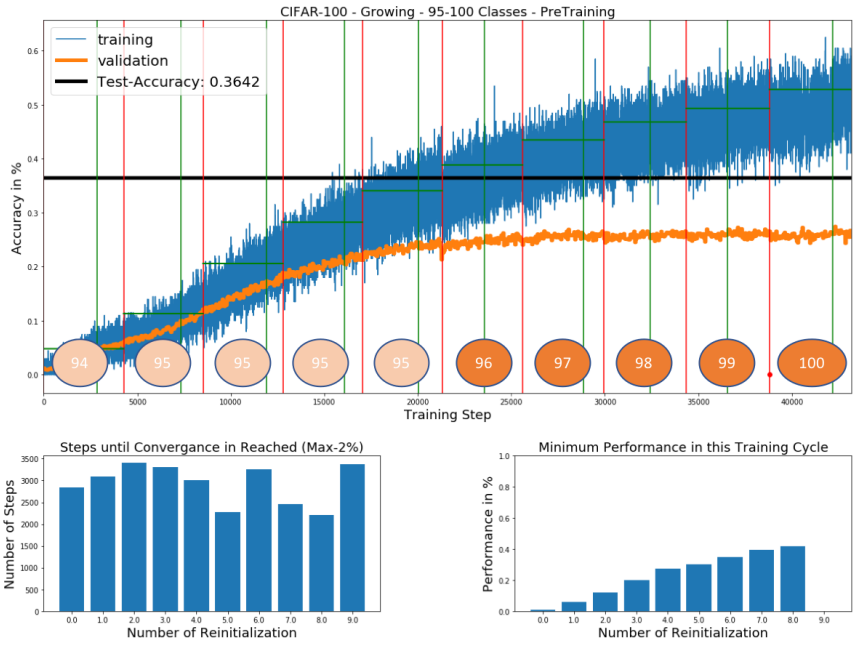

FIGURE 3.11: Training accuracies on CIFAR-100. Starting training with 95 classes and introducing one class each epoch until all 100 classes are in the training data. The network is continuously growing with each new class introduction.

A big drop in performance following the new class introduction does not take place. This, however, can be due to the small percentage the new class makes up in the training data since it only contributes a bit more than 1% to the overall accuracy. Now the accuracies on the new class and on the old classes will be examined.

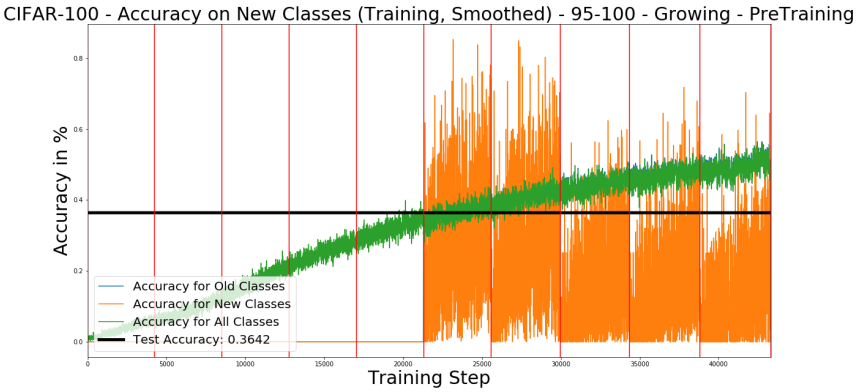

FIGURE 3.12: Training accuracies on CIFAR-100 for old (green) and new (orange) classes. Starting training with 95 classes and introducing one class each epoch until all 100 classes are in the training data. The network is continuously growing with each new class introduction.

In figure 3.12 you can see that the accuracy on the new class starts low and then continuously increases while the accuracy on the old classes stays relatively unchanged. This demonstrates that just as with CIFAR-10 a new class can be learned later on in the training without forgetting about the old classes. But what happens if more than one new class needs to learn?

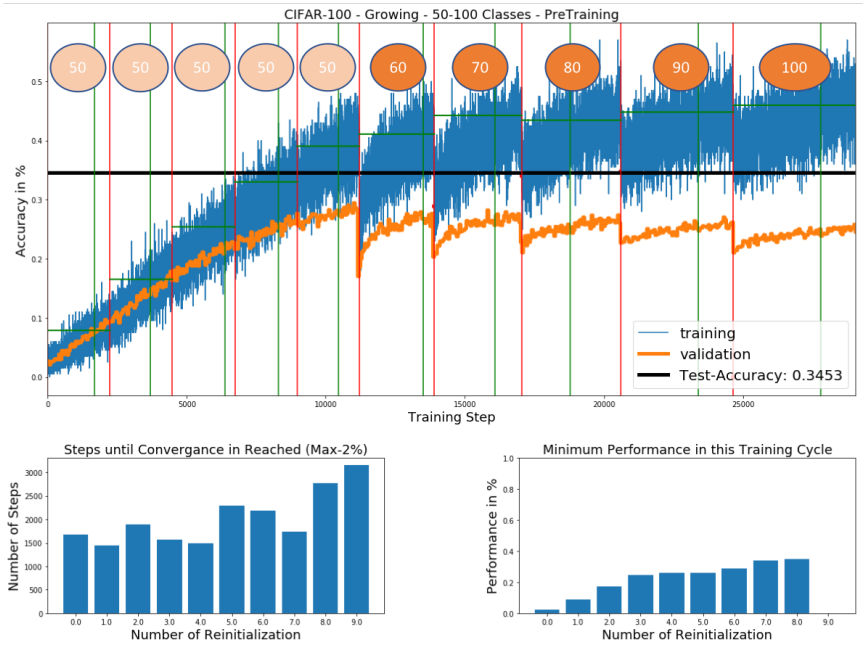

FIGURE 3.13: Training accuracies on CIFAR-100. Starting training with 50 classes and introducing ten new classes each epoch until all 100 classes are in the training data. The network is continuously growing with each new class introduction.

In figure 3.13 it can be seen that when introducing ten classes at once, the network performance still never drops below chance again. It drops a bit but since the new classes make up more than 10% of the training and validation set it is a natural effect. However, in figure 3.14 you can see that also the accuracy on the old classes drops a bit when ten new classes are introduced. This could be explained by false positives for the new classes in the beginning of training when the weights connecting to the new output nodes are still random.

FIGURE 3.14: Training accuracies on CIFAR-100 for old (green) and new (orange) classes. Starting training with 50 classes and introducing ten new classes each epoch until all 100 classes are in the training data. The network is continuously growing with each new class introduction.

Overall, it can be seen that no matter if one or ten classes are introduced the new classes do not drag the accuracies below chance and they can be learned after a bit of training. The condition with only one class introduction at once reaches a slightly

higher test performance than with ten classes at once. However, this difference could easily be compensated by longer training since in the ten class introductions the classes from 49 to 99 are seen less often by the network than in the one class introduction condition.

Now a network which is continuously learning for a very long time will be examined, starting with 50 classes and introducing one new class each epoch until all 100 classes are in the training data. This of course also requires a much longer training leading to a higher performance overall.

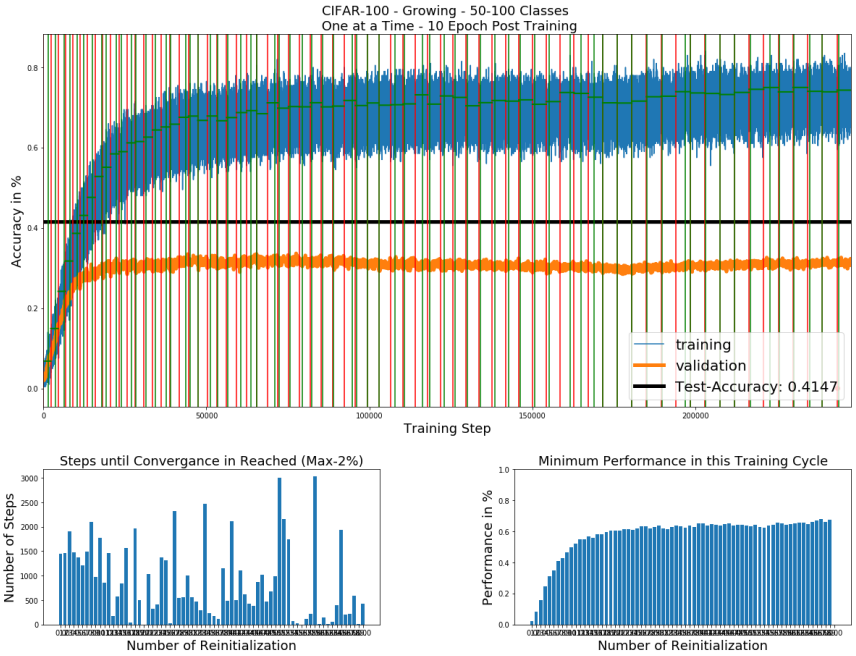

FIGURE 3.15: Training accuracies on CIFAR-100. Starting training with 50 classes and introducing one new class each epoch until all 100 classes are in the training data. The network is continuously growing with each new class introduction.

It can also be seen that the network performance does not suffer visibly from a new class introduction. Even when half of the classes are introduced at a later point of training and the last layer doubles in size the network still performs very well on the whole data set.

3.3.2 Condition Comparison for CIFAR-100

Now, a comparison will be made between the test results of the network training shown in 3.3.1 and A.3.1. Since computational limitations made it impossible to run these experiments 100 times, these are just the results of one run and cannot be quantitatively compared. However, one can already see that there are no discernable differences between the conditions (normal, growth, no growth) and the different number of classes that are introduced at once (figure 3.16).

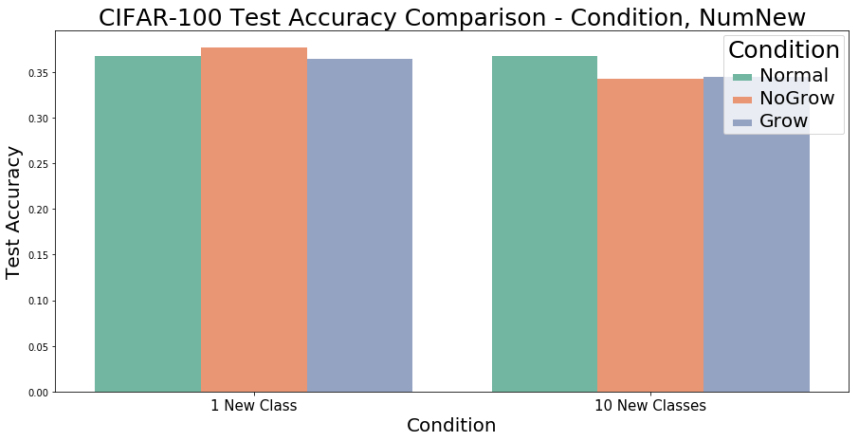

FIGURE 3.16: Comparison of test accuracies on CIFAR-100 under different training conditions.

In the condition comparison where only one class is introduced, all conditions seem to perform about equally well, the best performance of the non-growing condition can be explained with the variance in test accuracies already shown for CIFAR-10. In the training with introducing ten classes at once seems as if the two continuously learning conditions are a bit worse than the normal condition and the one new class condition. However, this could also be due to variances or length of training.

3.4 Weight and Gradient Distribution Change After Class Introduction

To investigate what a new class introduction does to a network, one must look at the weight distribution inside the network as well as its gradients. For simplicity, the last layer will be examined because this is where the size increase is happening and where one should see the biggest change if there is any.

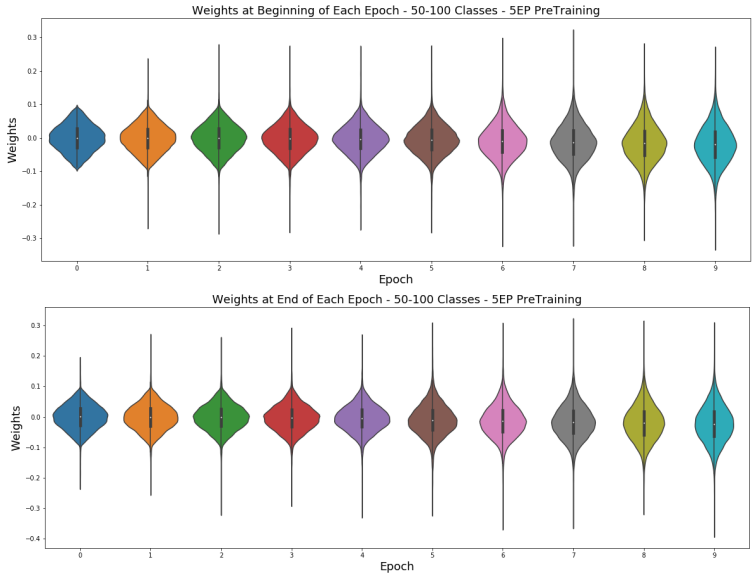

FIGURE 3.17: Distribution of weights in the last layer for each epoch. Network is trained on CIFAR-100 with five epochs of pretraining with 50 classes. Starting in epoch 5 ten new classes are introduced in each epoch until in epoch 9 all 100 classes are in the data set. The last layer of the network grows with the number of classes present in the data. (Top) Weight distribution at the beginning of an epoch. (Bottom) Weight at the end of the epoch.

In figure 3.17 we can see the distributions of weights in the last layer for each of the ten epochs of training. On the top one can see the weights in the beginning of an epoch (starting with epoch five this is right after a new class has been introduced). At the bottom the weights at the end of an epoch are displayed.

Overall there are no visible effects of new class introduction on the weight distributions. The weights always keep the same distribution and stay approximately in the range between -0.25 and 0.25.

In the appendix (figure A.11 and A.12) one can find the weights connecting to each individual output node. But since the weights to the newly initialized nodes are sampled from the old weights they already start out with the same distribution as all other weights.

3.5 Condition Comparison With Longer Training

3.5.1 Longer Training on CIFAR-10

In the previous plots the continuous learning conditions had a bit of a disadvantage since the classes introduced in the later epochs were seen less often than they were seen in the normal training condition. In the most extreme case of class nine each example of it is only seen once in the growing conditions since it is only introduced in the last epoch. In the normal condition in comparison each example of class nice has been shown five times to the network. This gives the normal network a strong advantage in identifying this class.

To compensate for that, I will now train the network under each of the three conditions for a longer time after the last class has been introduced. I tested different lengths of training. The distribution of test results can be found in figure 3.18. The individual training plots can be found in the appendix (A.2.3).

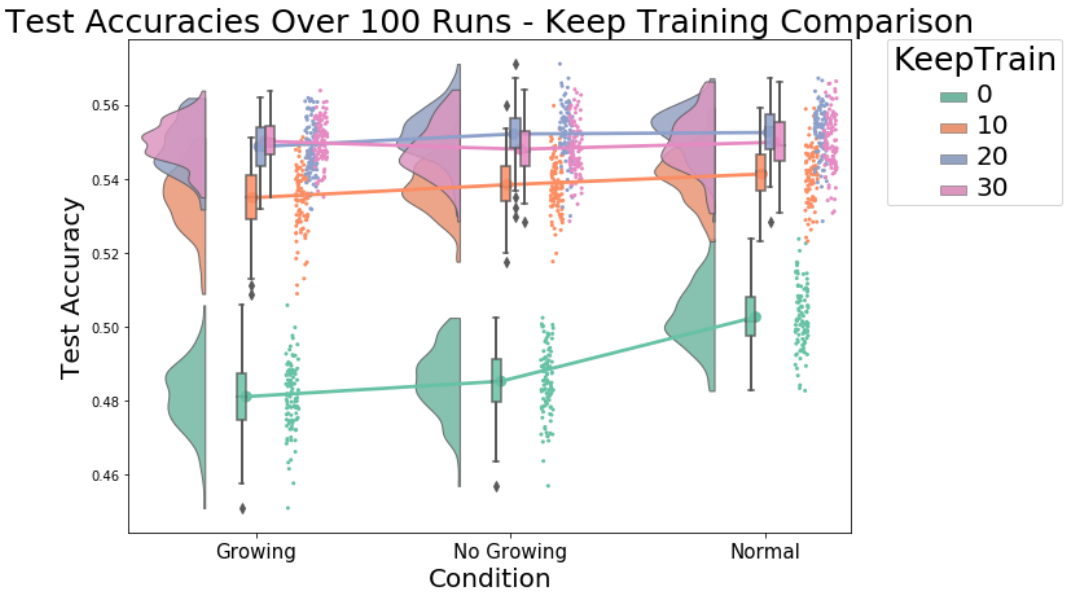

FIGURE 3.18: Performance comparison between all three conditions for different durations of training. KeepTrain=0 is the normal test performance as shown above, with stopping after ten epochs. KeepTrain X>0 means that the network was trained for X more epochs after the tenth epoch, which is the epoch in which the last class has been introduced.

One can see that all conditions continue to improve with more training. The two continuous learning conditions (left and middle) catch up to the performance of the normal network after about 20 epochs of post training. After thirty epochs, the performance does not increase anymore but rather decreases slightly. This indicates the set in of overfitting where the network models the training data too well and therefore gets worse at generalizing to the test data. In figure 3.19 you can see how the training performance keeps increasing while the validation (yellow) and test (black) performance saturate after approximately 20 epochs.

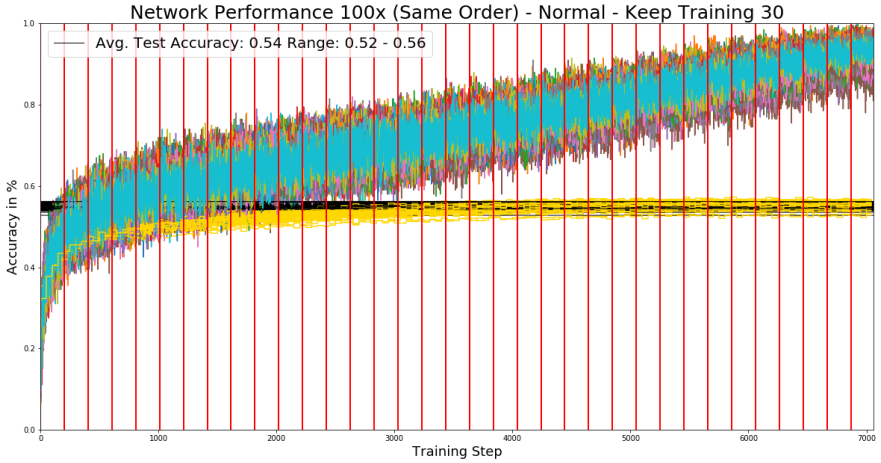

FIGURE 3.19: Performance of 100 runs of normal training on CIFAR-10. Yellow lines show the validation performance, black the test performance.

When testing the difference between all three conditions after 30 epochs of post training in an ANOVA, no significant differences can be found anymore, F(2,297), p=0.06. This indicates, that the initial lower performance of the iterative learning condition was caused by the lower exposure of the network to the classes introduced last. It shows, that a network can be trained with iterative class introduction during training and still reach a comparable performance to a network trained with all classes from the beginning on.

3.5.2 Longer Training on CIFAR-100

In figure 3.20 you can see that also the CIFAR-100 network profits from a longer training.

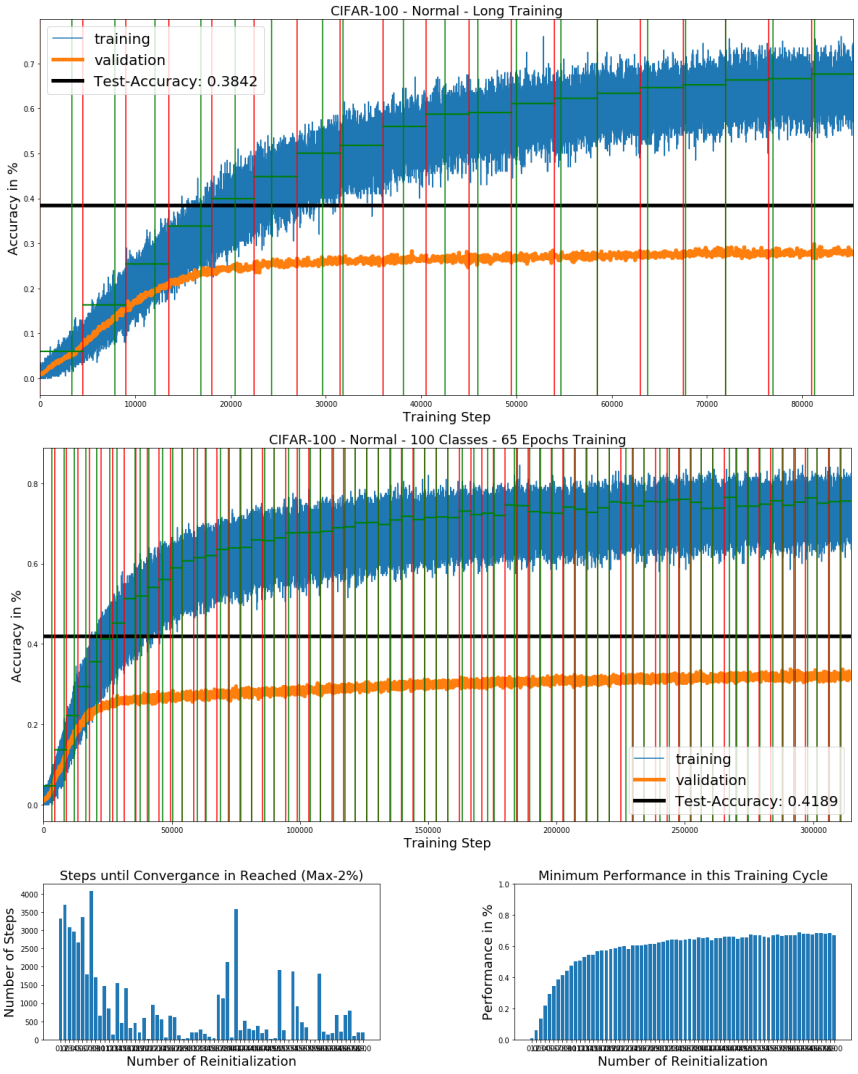

FIGURE 3.20: CIFAR-100 runs with longer training under normal conditions (No class introductions). (Top) 19 epochs of training. (Bottom) 65 epochs of training.

In figure 3.21 you can see that also a growing network profits from longer training and catches up to the performance of a normal network with longer training. This is consistent with the findings on CIFAR-10.

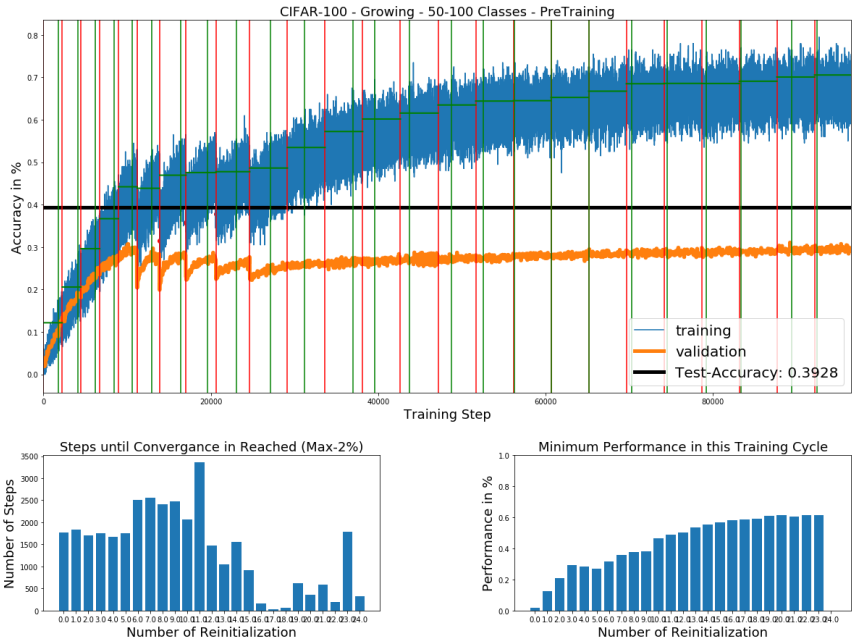

FIGURE 3.21: A CIFAR-100 run with longer training (25 epochs) on a growing network, introducing ten classes at once during epoch 5-10.

3.6 Catastrophic Forgetting for CIFAR-10

A network can learn a new class later during training without forgetting about the old classes. But, in all previous experiments the network still trains on all of the old classes. The problem of catastrophic forgetting in neural networks is that once a new task is presented to a network, it forgets all about the old task (Kemker et al., 2017). In this section, the network performance results will be examined when the old classes are underrepresented or not present at all anymore after a new class has been introduced.

The most extreme case first: After a new class was introduced all old classes are not shown anymore. The question is if the network will remember the old classes after learning the new class.

In figure 3.22 one can see a network training under such a condition. For the first two epochs the network is trained on six classes. In the third epoch a seventh class is added, and the first six classes are taken out of the training set. The validation data now contains all seven classes. In the next epoch only the eighth class is shown to the network with eight classes in the validation set, then the ninth and then the tenth class are added following the same scheme.

FIGURE 3.22: Performance of a growing network on CIFAR-10 over 100 runs. When a new class is introduced (starting after two epochs) only this class is shown, and the old classes are discarded from the training data. Validation and testing is performed with all classes that have been introduced by the time it is measured. Yellow lines show validation performances. Black lines show test accuracies.

One can see that when showing only one class the network's training performance quickly surfaces to a close to perfect performance on this class. However, the validation performance on all classes (yellow) decreases to a performance around chance. The average test performance on all classes at the end is at chance level (0.1). Figure 3.23 shows this difference between the performance on the old and new classes again. One can see that the performance on the new class (orange) is close to perfect while the performance on the old classes (blue) quickly drops.

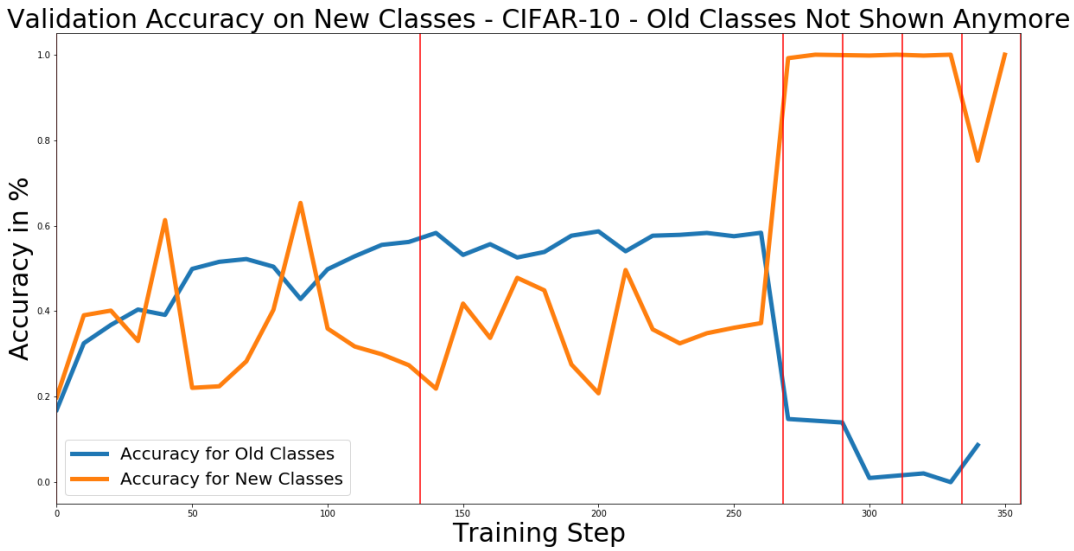

FIGURE 3.23: Performance of a growing network on CIFAR-10 of one run. Comparison between the accuracy on the new class (orange) and on the old classes (blue).

These results indicate that the network forgets about the old classes once they

are not in the training data anymore. The training on only the new class seems to destroy the previously learned weights for the old classes. The extremely good performance on that class hints that the network might just learn the data distribution instead of the actual object representation. The network seems to quickly figure out that it performs best when it only predicts this new class and adjusts its weights accordingly. Since the accuracy on the old classes drops close to zero it is probable that the performance of 0.1 on the test data comes from always predicting class nine, the last class that has been trained on.

Now, not showing the old classes anymore is a very extreme case. What would happen if the old classes are still shown every once in a while, but are a bit underrepresented in the data set? This means the new classes are shown more often but the network still should not completely forget about the old class to keep up a decent performance. Figure 3.24 shows different compositions of the data set. In the first row one can see a normal condition where the new class makes up 7-10% of the training data. In the next row the new class makes up 20% of the training set and the other 80% are old classes. The last two rows show network runs where the new class makes up 30 and 50% of the training set respectively. The validation and training set is always balanced between all introduced classes.

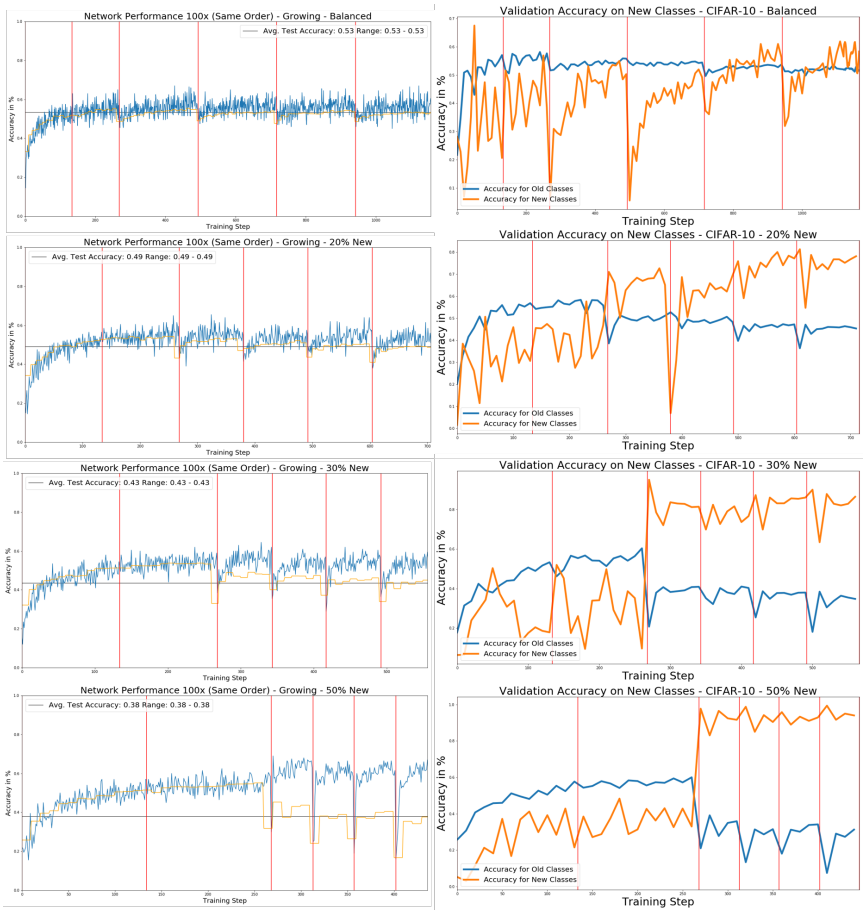

FIGURE 3.24: Performance of a growing network on CIFAR-10 of one run. (Top to Bottom) Comparison between different percentages in the training data that consist of the new class. From top to bottom: normal (7-10%), 20%, 30%, 50%. (Left) Performance of the network under the respective condition. (Right) Comparison between the accuracy on the new class (orange) and on the old classes (blue).

One can see that the overall performance gradually decreases (figure 3.24 and 3.25) the more unbalanced the training set gets. It therefore does not help to overrepresent the new class to learn it faster. An unbalanced training set teaches the network a tendency towards the over represented class and reduces the test and validation accuracies.

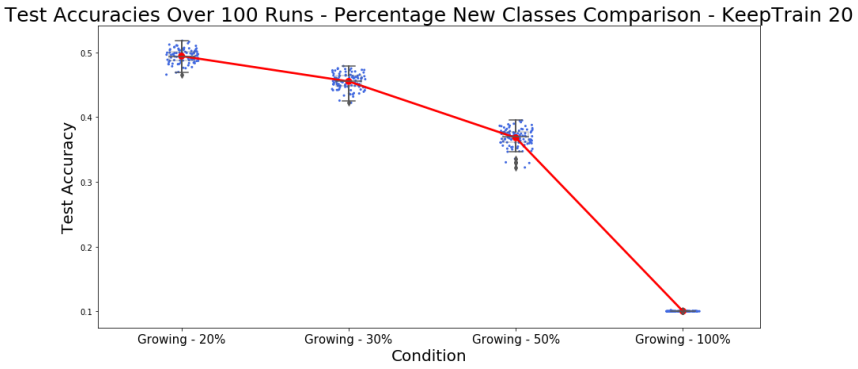

FIGURE 3.25: Performance of a growing network on CIFAR-10 of one run. Comparison between different percentages in the training data that consist of the new class. From left to right: 20%, 30%, 50%, 100%.

3.7 Catastrophic Forgetting on CIFAR-100

Especially when introducing more than one class at once it might seem beneficial to show all the new classes a bit more often than the old ones to learn them faster. To investigate this and the question of how catastrophic forgetting impacts bigger data sets, a growing network will be trained with different new-old class ratios.

In figure 3.26 one can see how the test accuracy gets lower the more unbalanced the data set gets. Just as with CIFAR-10, it does not seem to help to show the new classes more often but rather decreases the performance of the network on a balanced test data set. Also, the network overfits a lot on the over-represented class and gets worse on the under-represented classes.

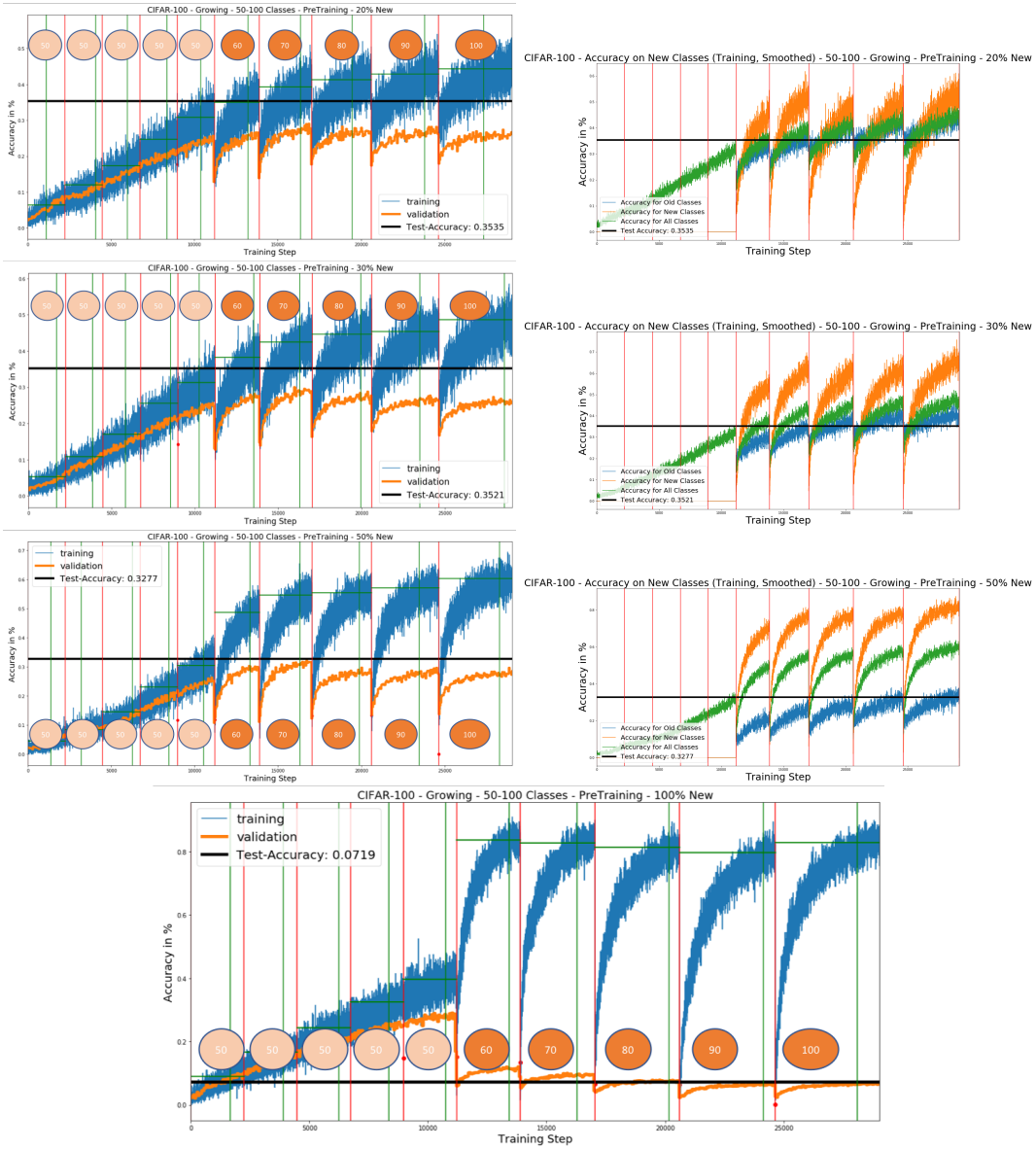

FIGURE 3.26: CIFAR-100 training with a growing network. Five epochs of pretraining on 50 classes, then introduce ten new classes each epoch until all 100 classes are in the data set. After five epochs trained with different old-new class ratios: 20%, 30%, 50%, 100% new classes (from top to bottom).

3.8 Freezing the CNN Layers for CIFAR-10

To investigate if the convolutional layers learn features which are universal enough to help classify new classes the network performance will be tested by freezing the convolutional layer after an initial training. This means that the CNN is only trained in the first epoch and on the first six classes. After the seventh class is introduced the two convolutional layers do not get weight updates anymore and only the fully connected layer keep training.

Figure 3.27 shows a comparison of the performances in the three conditions for networks that were frozen after the first epoch (grey) and networks that were not

frozen (green). The corresponding performance plots can be found in the appendix (A.9).

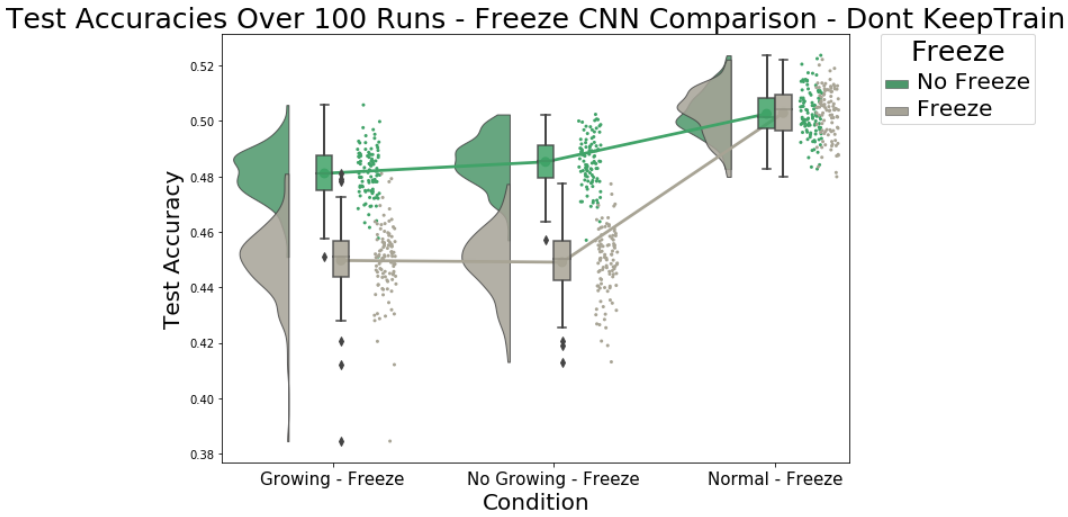

FIGURE 3.27: Performance comparison of networks trained with freezing (grey) and without freezing (green) in all three conditions (CIFAR-10).

You can see that the networks which were frozen perform overall worse than the unfrozen networks in the two continuous learning conditions. For the normal training condition, no clear difference can be observed. The normal condition has the advantage that in the first epoch all classes are already present in the training data. Therefore the convolutional layer can be trained on all classes, at least in the first epoch. In the two other conditions however, the convolutional layers are never trained on the last four classes.

In section 3.5 I have shown that the continuously learning networks actually need some more training to reach their full potential. In figure 3.28 you can see how the performance in the two continuously learning conditions improve with longer training on a frozen network while the normal condition does not increase by much.

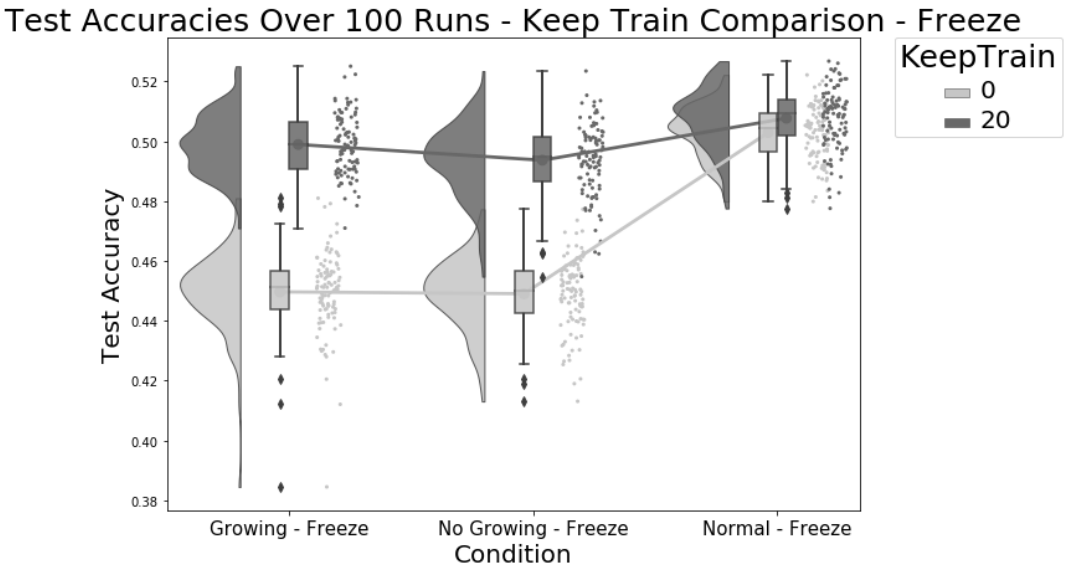

FIGURE 3.28: Performance comparison of networks trained with freezing between short (5 epochs) and longer training (25 epochs) in all three conditions on CIFAR-10.

When now comparing the longer training on a frozen network to a network that was trained for longer without freezing (figure 3.29) we see that now all conditions have a worse performance when training with frozen convolutional layers. The continuous learning conditions on a frozen network are still a bit worse than the normal condition but also the normal condition performs worse when frozen. There is a significant difference between the performance of frozen networks and fully trained networks, $F(1, 4246.76)$, $p<0.001$. Also the difference between the three conditions is significant here, $F(2, 33.6)$, $p<0.001$. Therefore freezing the convolutional layer of the network after the first epoch decreases its performance significantly. It also leads to a bigger difference between the conditions with the normal condition being the best and the growing condition the worst. A growing network seems to be affected the most by the layer freezing since here the layer is already frozen before all nodes and stimuli have been present.

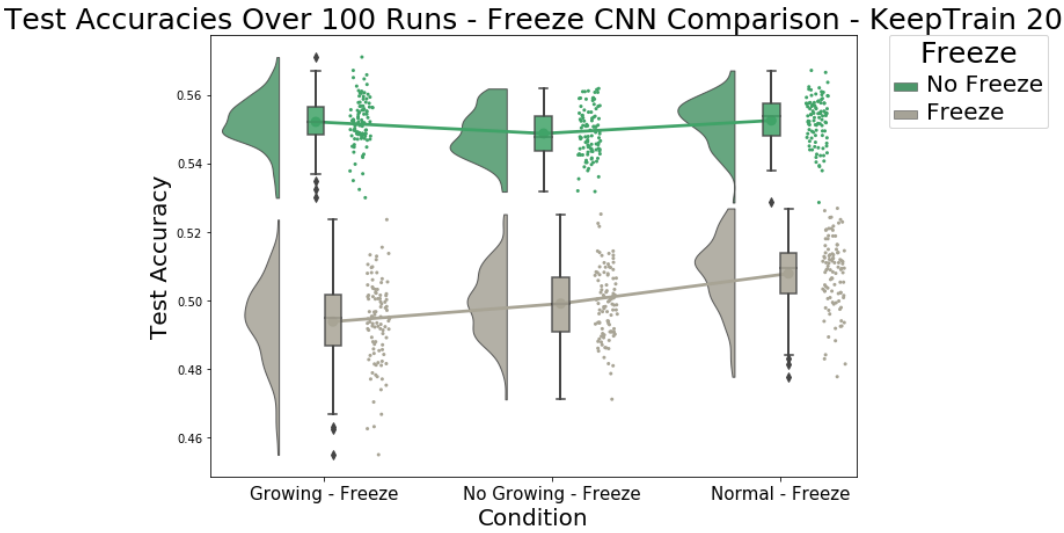

FIGURE 3.29: Performance comparison of networks trained with freezing (grey) and without freezing (green) in all three conditions (CIFAR-10) with 20 epochs of post training after the last class introduction.

These results demonstrate, that the convolutional layers are still used to finetune the network and help improve its accuracy. Since big differences cannot be found between freezing and non-freezing in the beginning of the normal training it seems as if the convolutional layers are only finetuned later on in the training to directly match the classes present in the training data. Therefore, it is not clear if the convolutional layer actually learns universal features or if it just starts to overfit to the categories in the data set. Since the continuous learning conditions already have a lower performance in the freezing condition compared to the fully trained network early on and they continue to have a lower performance after more training, even lower than the frozen normal training, it seems as if the learned features are not very universal and fail to generalize well to the newly introduced classes.

However, the performance of the frozen networks is not altogether bad and still way above chance for all conditions. Figure 3.30 shows that this performance mainly originates from the fully connected layers since the frozen performances are only slightly better than no CNN training overall. In the no CNN training condition, the convolutional layers get randomly initialized at the beginning of training and then do not receive any updates. The fully connected layers then learn quite well to classify the randomly manipulated inputs.

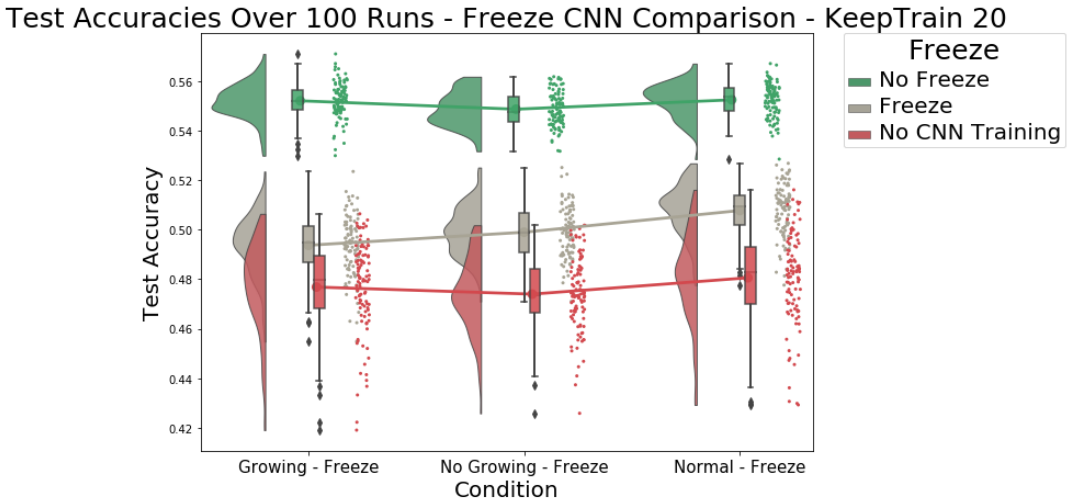

FIGURE 3.30: Performance comparison of networks trained with freezing (grey), without freezing (green) and with no CNN training at all (red) in all three conditions (CIFAR-10) with 20 epochs of post training after the last class introduction.

Overall it seems as if the convolutional layers only help a little bit with the classification of the stimuli. They increase their contribution to the performance after a longer training (compare figure 3.27 and figure 3.29). This can be observed well when looking at the normal performance in the four figures above. After only short training the frozen and unfrozen performances are almost the same (figure 3.27). When training for longer, the frozen performance does not increase by much (figure 3.28). The normal performance without freezing still increases with longer training (figure 3.29 and 3.30). This indicates that in the later phases of training more performance gets squeezed out of the convolutional layer while in the beginning most of the performance results from the fully connected layer.

However, the features learned in the convolutional layers during the first epoch still contribute to the performance and the shortly trained convolutional layers (grey) are significantly better than the networks with no CNN training at all (red), $F(1,375.53)$, $p<0.001$. Also, the difference in the conditions is less pronounced in the no training condition ($F(2,3.69)$, $p=0.03$) than in the short CNN training (freeze) condition ($F(2,42.46$, $p<0.001$). Since in the freezing condition the continuously learning networks are worse than the normal network, it indicates that the convolutional layers learn features specific to the first six classes in the first epoch but can't generalize as well to the new classes. The normal training can deal better with the layer freezing since they have already been trained on all ten classes.

3.8.1 Freezing the CNN Layers to Prevent Catastrophic Forgetting

One idea to prevent catastrophic forgetting could be to freeze part of the network so that information encoded in that part does not get lost after the old classes are not seen anymore. Figure 3.31 shows the performances of a growing network with an unbalanced training set (30% new classes) while freezing the CNN after the first epoch on the right compared to not freezing it on the left. There is no discernable difference between the two.

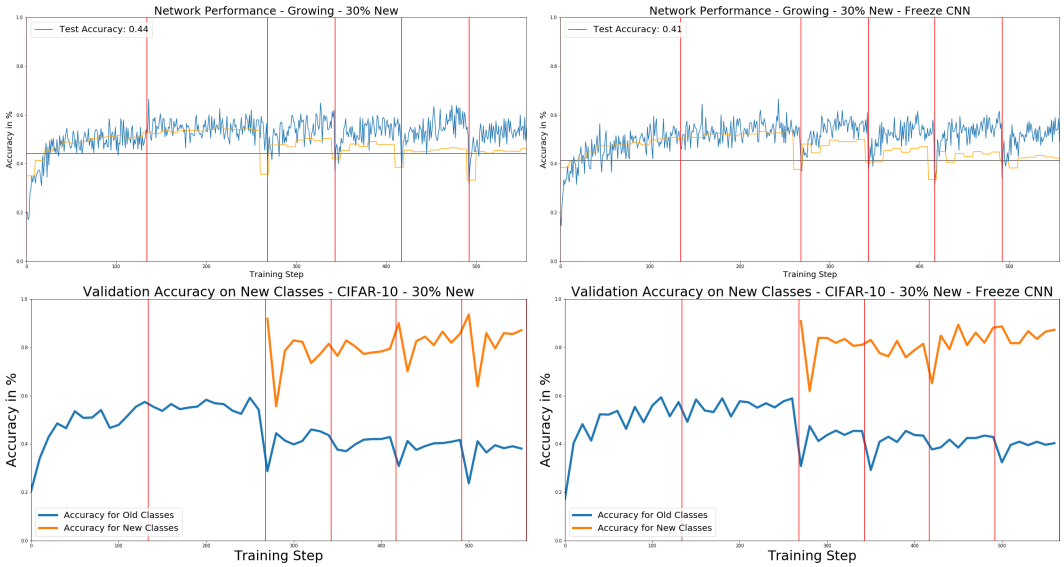

FIGURE 3.31: Performance comparison of networks trained with freezing (right) and without freezing (left) while growing on CIFAR-10. (Top) Training-, validation-, and test accuracies over time. (Bottom) Comparison between validation accuracy on new (orange) versus old (blue) classes.

In figure 3.32 you can see that freezing the two convolutional layers does not help with catastrophic forgetting. The performance of the frozen network is significantly lower than for the fully trained network, $F(1, 68.21)$, $p<0.001$. Freezing the convolutional layer therefore does not help to preserve information about the old classes for longer if they are not shown that often anymore.

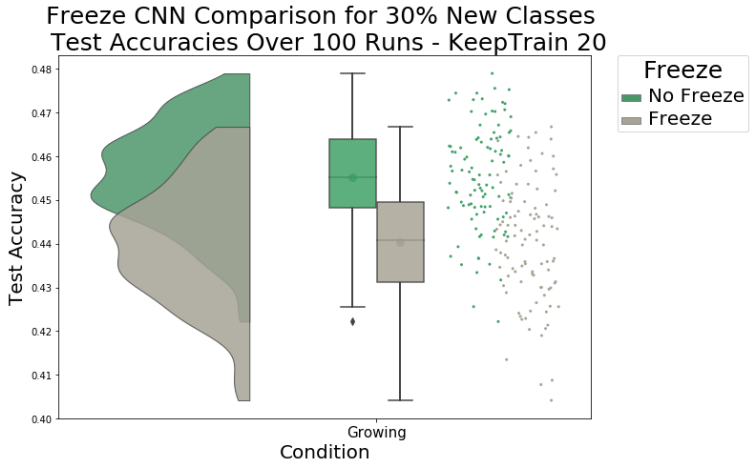

FIGURE 3.32: Performance comparison of networks trained with freezing (grey), without freezing (green) with the new class being over represented (30%). Network trained while growing with 20 epochs of post training after the last class introduction on CIFAR-10.

3.9 Class Difficulty Differences

During the previous experiments it became evident that the network did not always reach the same accuracy on a new class that was introduced. This effect was investigated and a difference was found in the difficulty of the different classes in the data sets.

3.9.1 Differences in All Data Sets

To investigate this, the performance of the network on each class during a normal training will be plotted separately.

FIGURE 3.33: Training performances on individual class of the CIFAR-10 data set during normal training on all classes simultaneously. Training performances are smoothed with a five-degree Gaussian for better visibility.

In figure 3.33 one can see the network performance on each of the ten classes in CIFAR-10. The network is trained on all classes at once and each class is shown equally often. Nevertheless, the network classifies some classes better than others. In this data set, classes depicting living beings (with the exception of horses and frogs) seem to be harder to classify for the network than other classes such as airplanes or Trucks. This can however also be a bias in the selection of representative pictures for each classes or other factors such as visual similarities between cats and dogs making it harder to distinguish them. Disregarding the source, there are definitely some differences in the difficulty of a class for the network.

These differences could also be found in CIFAR-100 and MNIST (figure 3.34). Especially in CIFAR-100 (figure 3.34, Top) one can see big differences in the network performance on the different classes.

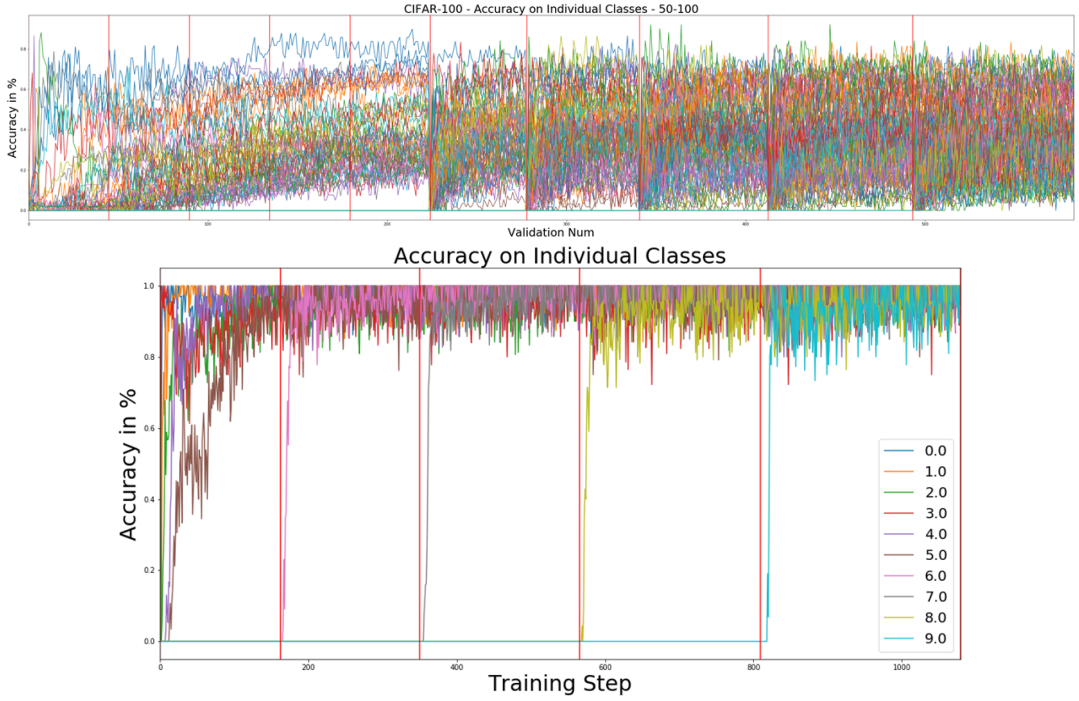

FIGURE 3.34: Training performances on individual classes of the CIFAR-100 data set (Top) and the MNIST data set (Bottom) during normal training on all classes simultaneously. Training performances are smoothed with a five-degree Gaussian for better visibility.

3.9.2 Variance Due to Class Order in CIFAR-10

So far, all continuous learning experiments have introduced the classes in the same order starting with class zero to five, then adding class six, then class seven and so on. Figure 3.35 shows how the variance in test accuracies increases when introducing the classes in a random order in each of the 100 runs.

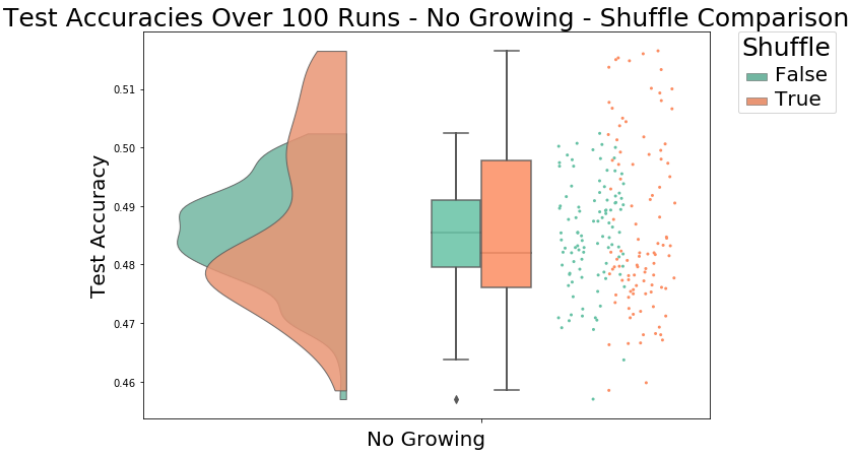

FIGURE 3.35: Distribution of test performances of a continuously learning network (no growing) with shuffled class order (orange) compared to fixed class order (green). Trained on CIFAR-10, no post training after the fifth epoch.

One can see that the order (0-5)-6-7-8-9 is on average (vertical line in the box plots) a bit better than a random order. However, there are some runs in the random

condition which reach a higher accuracy than any of the (0-5)-6-7-8-9 runs. To investigate these well-performing runs one should first examine the order in which these runs had their class introductions.

Figure 3.36 shows in which order classes were introduced for the runs which reached a test accuracy above 0.5. Each row represents the order for one run. The first six columns show the classes which were present in the first epoch. The seventh column shows which class was introduced in the second epoch, the eighth column the new class in the third epoch and so on. You can already detect some patterns in this. For example, the last column (representing the class that was introduced last) contains only classes 2,3,4 and 5. When you refer back to figure 3.33 these are all classes that the network had an accuracy below the average. Therefore, it seems as if all runs that reach a good test accuracy prefer to show the harder classes in the end of the network training.

```
([[4, 6, 7, 2, 1, 8, 9, 5, 0, 3],
  [8, 0, 1, 4, 6, 5, 7, 2, 9, 3],
  [4, 2, 0, 9, 6, 8, 1, 7, 3, 5],
  [7, 2, 3, 8, 5, 9, 0, 1, 6, 4],
  [8, 1, 0, 2, 4, 9, 6, 7, 5, 3],
  [6, 4, 1, 8, 3, 7, 0, 9, 2, 5],
  [6, 4, 9, 7, 0, 5, 1, 8, 3, 2],
  [8, 2, 4, 7, 9, 6, 1, 0, 3, 5],
  [4, 3, 7, 1, 0, 9, 2, 6, 8, 5],
  [7, 4, 1, 9, 6, 5, 2, 8, 0, 3],
  [8, 2, 1, 5, 0, 9, 7, 4, 6, 3],
  [7, 5, 2, 8, 6, 1, 4, 0, 9, 3],
  [2, 9, 4, 8, 1, 6, 7, 0, 5, 3],
  [9, 1, 6, 7, 4, 5, 2, 8, 0, 3],
  [0, 7, 1, 6, 8, 9, 5, 3, 2, 4],
  [7, 6, 1, 4, 8, 9, 5, 0, 2, 3],
  [0, 2, 1, 5, 9, 3, 8, 7, 6, 4],
  [7, 2, 4, 6, 1, 3, 8, 9, 0, 5],
  [1, 7, 4, 8, 9, 0, 5, 6, 3, 2],
  [0, 1, 4, 6, 8, 7, 9, 5, 2, 3],
  [1, 9, 3, 8, 0, 4, 7, 6, 5, 2]])
```

FIGURE 3.36: List of orders of class introductions for all runs with an accuracy > 0.5. Each row shows the order for one run. The first six numbers of a row represent the classes that were present in the first epoch. The next four numbers show in which order the new classes were introduced each epoch.

To investigate this further figure 3.37 now shows the relationship between the position in which a class was shown in the best runs in relation to their difficulty. To measure class difficulty, I use the accuracy of the network on this class during normal training. These accuracies are extracted at the end of a normal training after five epochs (figure 3.33). So, a lower accuracy corresponds to a higher-class difficulty. The network accuracies for a class are shown on the abscissa of figure 3.37. The ordinate displays the position of a class in the best class orders. Each point in the plot therefore displays a position-class accuracy combination from the list shown in figure 3.36.

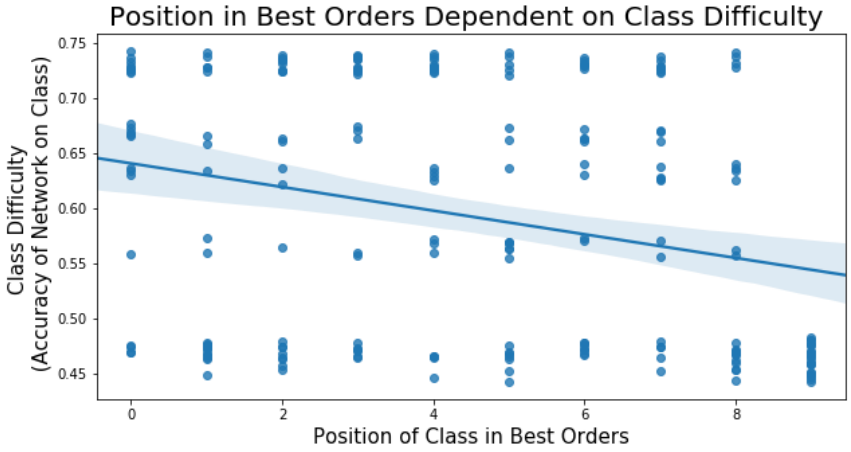

FIGURE 3.37: Orders of class introductions for all runs with an accuracy > 0.5 mapped on their class difficulties. Class difficulty is measured by the accuracy of a normal network on this class.

One can now see the trend again which was already observed in the raw list of class orders. The classes in the last position of the best performing class orders all have a high difficulty i.e. a low accuracy of the network. When fitting a regression line one can see that there is a negative correlation of -0.27 (Pearson correlation) between the position of a class in a good performing class order and a normal networks performance on this class, $F(1,16.43)$, $p<0.001$. This means that the runs which reached an accuracy above 0.5 tend to introduce the easy classes first and the harder classes last.

3.9.3 Curriculum Learning on CIFAR-10

From the previous investigation of the best performing runs we have seen that these runs tend to show easy classes first and hard classes last. This behavior has previously been reported in Bengio et al., 2009 and first proposed by Elman, 1993. They state that neural networks can reach a higher performance if they first train on easy examples before they see harder ones.

To test this theory, the two extreme opposite conditions were compared. The first, where all classes are sorted from easiest to hardest and are shown in that order and the second where all classes are sorted from the hardest to the easiest.

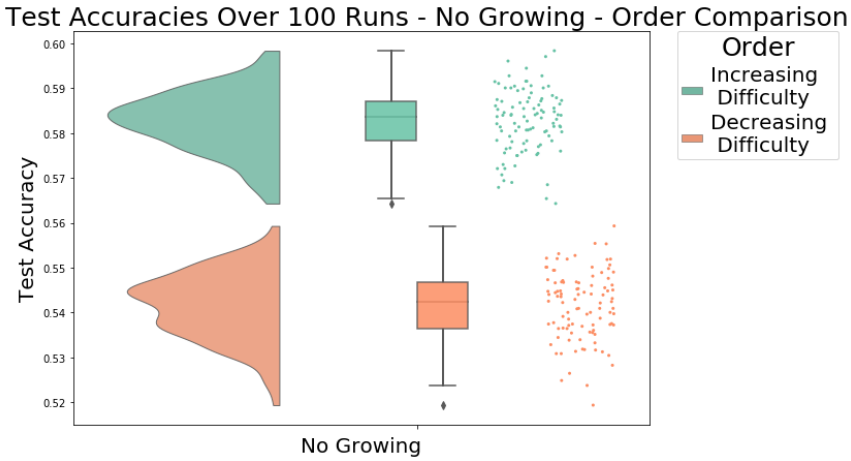

FIGURE 3.38: Test accuracies of 100 runs with decreasing class difficulty (red) compared to increasing class difficulty (green) on a continuously learning network (no growing) of CIFAR-10 with 20 epochs of post training.

In figure 3.38 one can see that there is a highly significant difference between the networks trained with classes introduced with increasing difficulty compared to decreasing difficulty, $F(1, 1586.68)$, $p<0.001$. Networks which train on the easy classes first before they see the harder classes reach a higher test performance than networks trained on hard classes first before they see the easier classes.

Interestingly one can see in figure 3.39 that the network trained with learning scheduling following a schedule which introduces hard classes last even reached a significantly better performance than a normal network trained on all classes for the same amount of time, $F(1,953.43)$, $p<0.001$. This means that when using a learning schedule and starting training with a few easy classes while gradually introducing the harder classes can lead to an overall performance improvement.

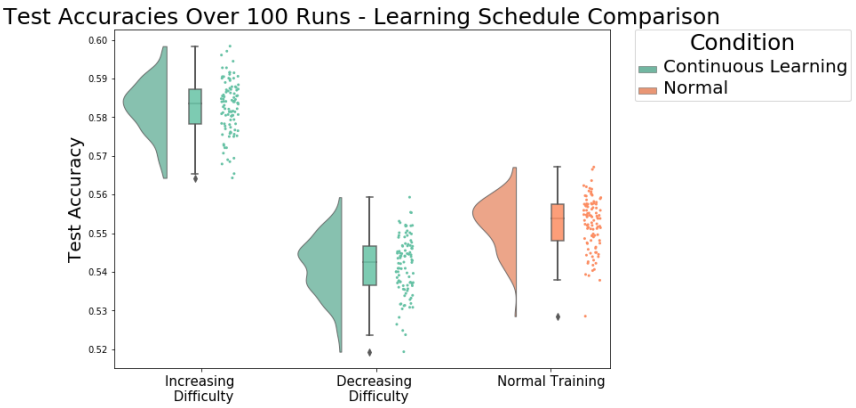

FIGURE 3.39: Test accuracies of 100 runs comparing learning scheduling with increasing class difficulty and decreasing class difficulty (green) on a continuously learning network (no growing) to a normally trained network. All three networks are trained on CIFAR-10 with 20 epochs of post training.

4 Conclusion

Overall, learning of new classes at a later point of network training without forgetting the previously learned knowledge is possible. To do this one can either initialize a big output layer or have the output layer grow dynamically. Both techniques lead to comparable network performances and do not fall short of a training with all classes from the beginning on, if given sufficient time.

Continuous learning needs more epochs to reach an equal performance as a normal trained network. However, this can be explained by the network's lower exposure to the classes introduced at a later point in time.

Continuous learning was also possible to perform on a larger data set (100 classes) and even the introduction of multiple new classes at once is possible without forgetting the old knowledge. Overall no significant differences in performance could be found between networks trained with all classes in the beginning and continuously learning networks after training for a sufficient amount of time.

When keeping already trained weights, the network performance does not drop below chance after a new class introduction. The performance drop is mainly influenced by the newly introduced class which is unknown to the network and needs to be learned. The performance on the old classes only drops marginally. When introducing multiple new classes, the decrease in accuracy on the old classes is only slightly bigger. Also, the distribution of gradients and weights of the network stays the same when introducing new classes. Therefore we can say that it is possible to learn new classes on the fly with relatively little impact on the network and its performance. Training time may be increased due to the lower exposure of the network to recently introduced classes.

The underrepresentation of new classes can however not be compensated by unbalanced training sets. When showing the new class more often than all other classes, the network becomes a victim of catastrophic forgetting and the accuracy on the old classes decreases significantly. In the most extreme case where only examples of the new class are shown, the old classes are forgotten completely after only a few steps. Also, in unbalanced data sets where the new class makes up for a higher fraction of examples in the trainings data than it naturally would, the overall network performance decreases. The network rather learns the statistical properties of the data set and starts overfitting the new class, reaching an unusually high performance on it while disregarding errors on the old classes. Also, when freezing part of the network to preserve learned information, the same effect occurs, and no performance gain can be accomplished. New class introductions should therefore always be accompanied with even amounts of old examples to avoid catastrophic forgetting.

Freezing the two convolutional layers of the network has shown that in the beginning most of the performance is achieved by the two fully connected layer. Later on, finetuning of the convolutional layers can lead to further performance increase. The features learned in the convolutional layer in the beginning of network training do not generalize well to new classes. They are only slightly better than random initializations but still more specific to the classes they have been trained on so that they generalize worse on new classes when learning continuously. In order to

reach best performances, the convolutional layer should not be frozen when working with class introductions during network training.

At last, there are some differences in the difficulty of individual classes in the data set. The results show that the order in which the classes are introduced matters for the final network performance on the test data set. Further investigations revealed that networks can reach the best performances if training begins with a subset of the easiest classes and the harder classes are gradually introduced during training. With this procedure even the conventional network training on all examples at once can be excelled.

It requires further investigation if the same principles apply to tasks of completely different nature, not involving image classification. Also, within image classification it is not completely certain if these principles always apply. However, being able to demonstrate these results on three different data sets makes it highly likely. A bigger concern would be how the network structure influences the results and if another network might work under different principles. Even though other results in the literature such as Bengio et al., 2009 and Graves et al., 2017 suggest that at least some of the findings also apply to different tasks and network structures this is a point that needs to be further addressed.

Using this method to overcome catastrophic forgetting did not completely work in the sense that training on the new class needs to be accompanied with a balanced set of old class examples. However, the argument could be made that in today's age this does not pose a significant challenge and it is still much faster than retraining the network completely from the beginning. Therefore it is possible to claim that this is a very simple and useful solution for adding new knowledge to an already trained network and continuously improving upon it. The argument that all data, even the old examples, needs to be stored does not appear to be a problem in most applications and should be regarded as a good practice in any case. Additionally, the new data acquired after deployment will most likely also contain new examples of old classes which can be used to further refine these classes while introducing a new class without having to fall back on the old examples. Besides that, a training involving old and new classes is closer to reality and to the way humans learn and explore. If one has learned the concept of a table, it does not mean that he would never come across a table again. Due to these considerations, the method introduced here provides a very simple, but practical solution of maneuvering around the problem of catastrophic forgetting.

Experiments on freezing layers have shown that the use of the convolutional layer is still not optimal, and it could be expected that if the learning of more general features could be archived, the overall network performance on new classes could also increase. This can be a point of further investigation. Also, how training on super classes before introducing sub classes is a promising idea to gain more insights and possible achieve higher network performances or even understanding of the data set.

Overall these results are very promising for the development of networks which need to keep learning while they are being used. This is the case in most real-world scenarios and the possibility to use this procedure to easily integrate new classes by increasing the network size can almost always be of advantage. Even though it is nice that with an optimized learning schedule more performance can be obtained from a network this can usually not be applied to real world scenarios where one cannot determine in which order classes are added to the data set and what their difficulty is. However, the procedure can be used for offline training of networks with the additional effort of obtaining a difficulty rating for each class first. For reaching

or exceeding benchmarks and deploying an optimal model this additional effort can easily be worth it.

A Appendix

A.1 Optimizer Comparison

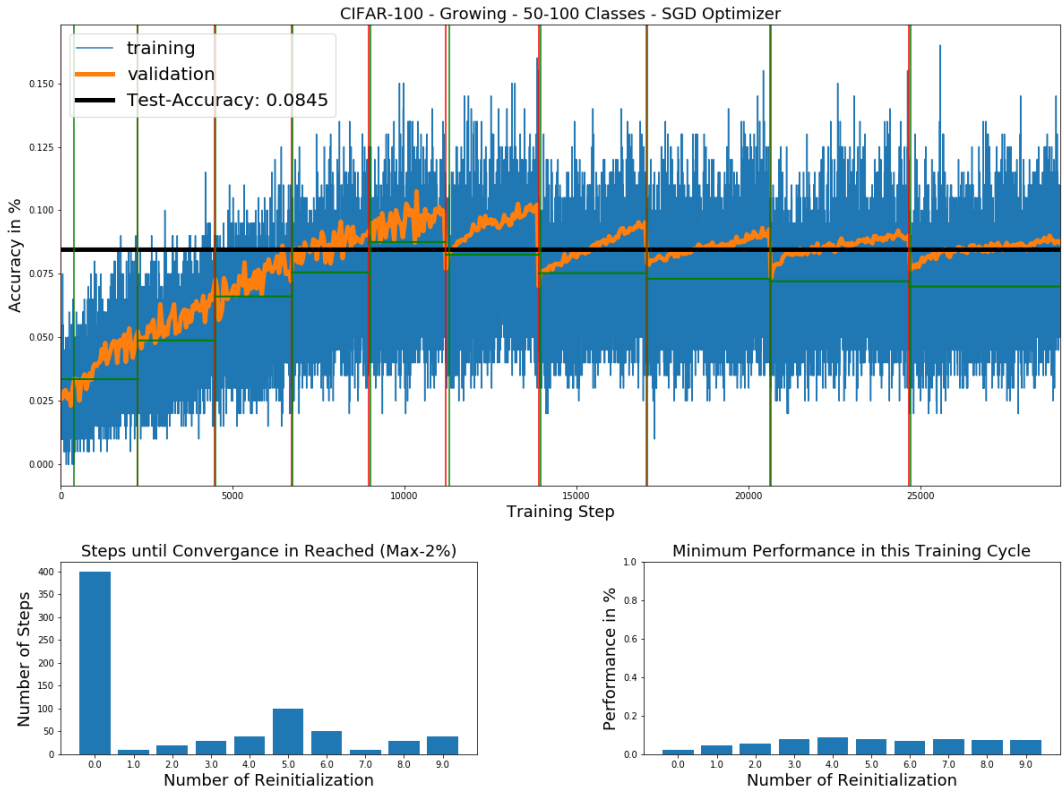

FIGURE A.1: Network growing on CIFAR-100 (50-5*10) using standard gradient descent.

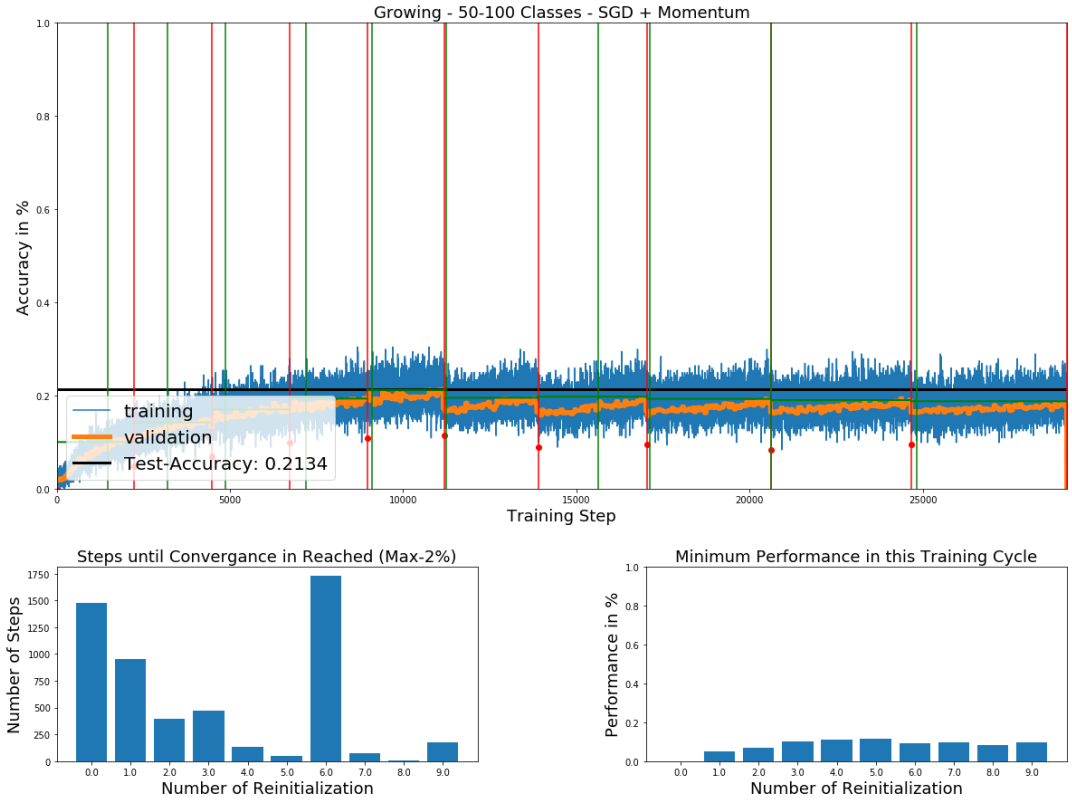

FIGURE A.2: Network growing on CIFAR-100 (50-5*10) using gradient descent and momentum.

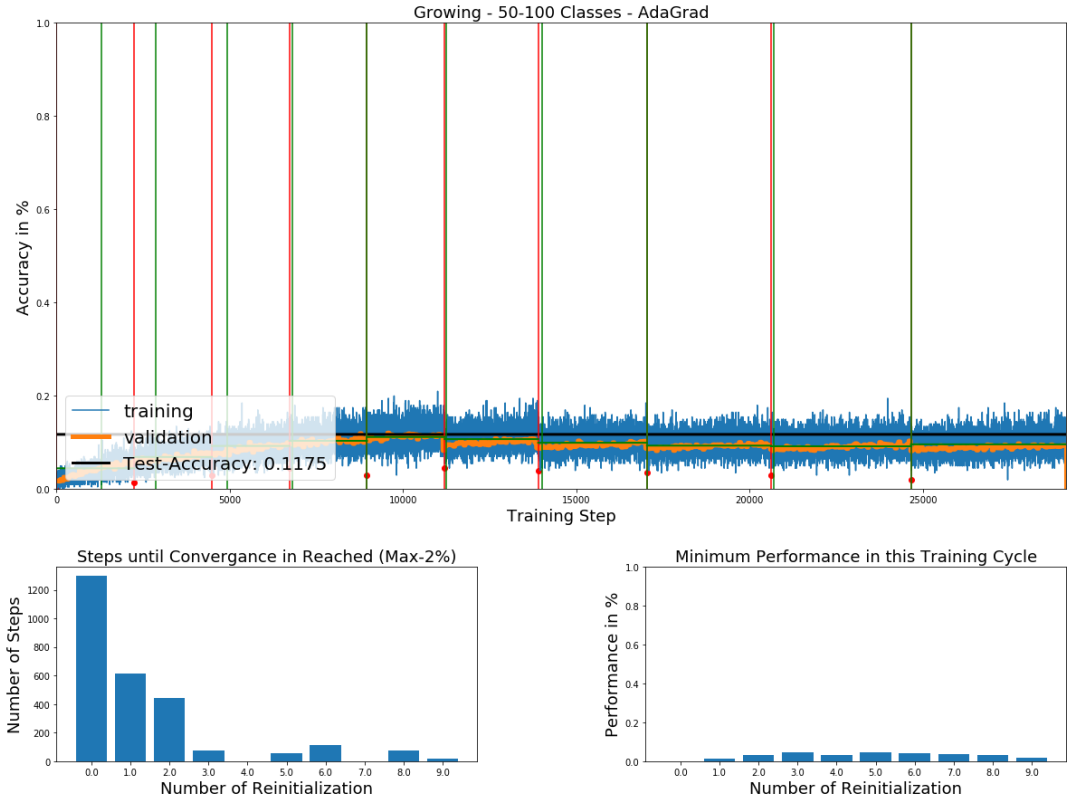

FIGURE A.3: Network growing on CIFAR-100 (50-5*10) using AdaGrad.

A.2 CIFAR-10 Experiments

A.2.1 Old-New Accuracy Comparison for a not Growing Network

FIGURE A.4: Comparison of accuracy's for old and new classes on CIFAR-10 with class introduction and no network growth.

A.2.2 Old-New Accuracy Comparison for a Growing Network - Not Smoothed

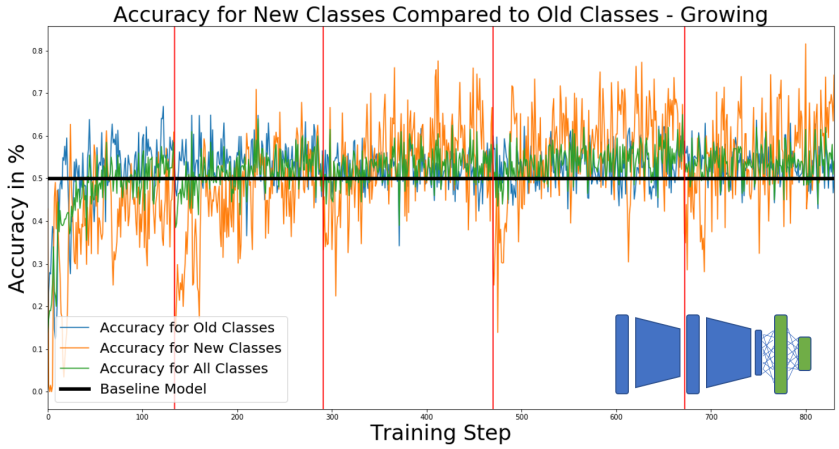

FIGURE A.5: Comparison of accuracy's for old and new classes on CIFAR-10 with class introduction and network growth. Accuracy's have not been smoothed.

A.2.3 Longer Training on CIFAR-10

A.2.4 Normal

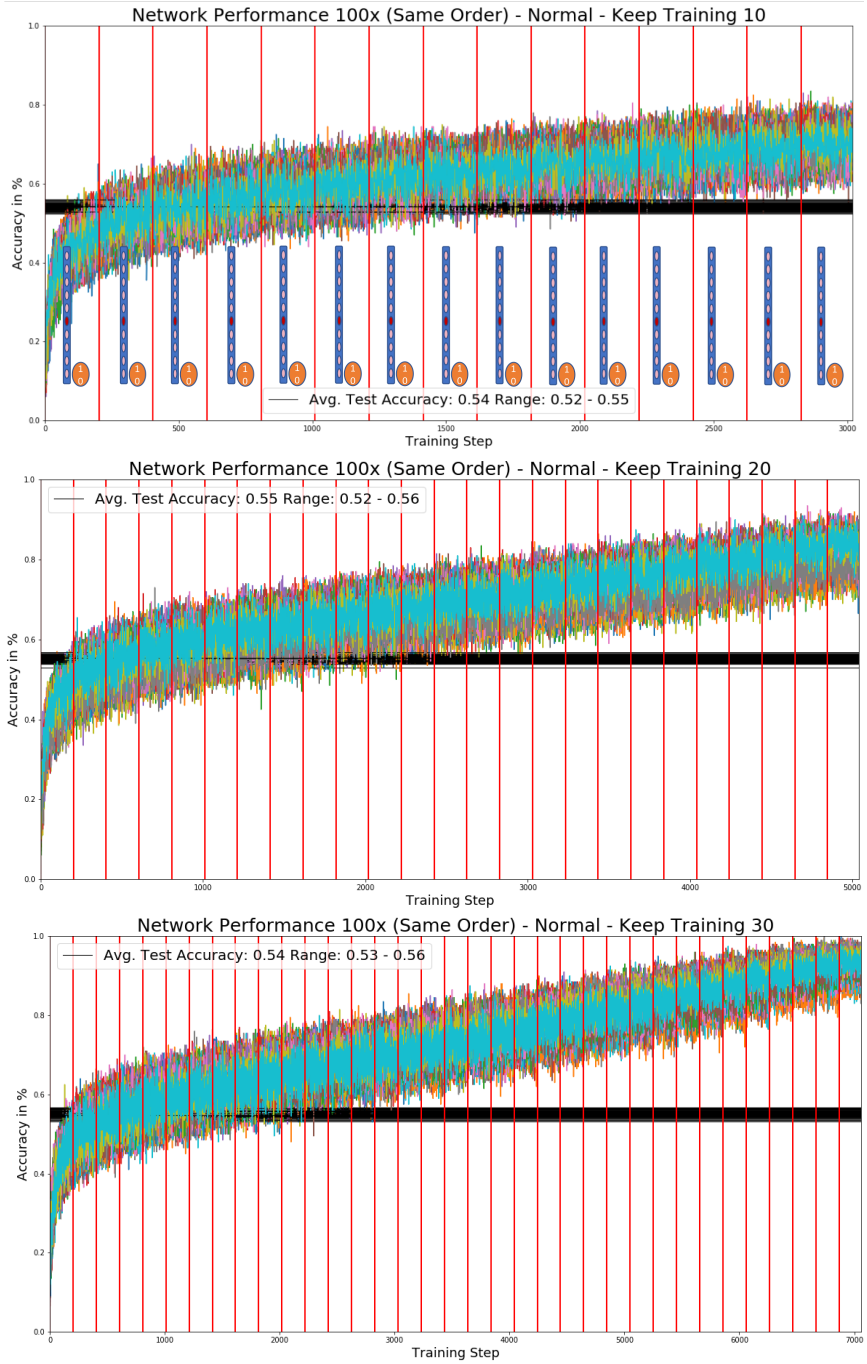

FIGURE A.6: Network accuracy's on CIFAR-10, normal training. (top) Train for ten more epochs after the last class has been introduced. (middle) Train for 20 more epochs after the last class has been introduced. (bottom) Train for 30 more epochs after the last class has been introduced.

A.2.5 No Growing

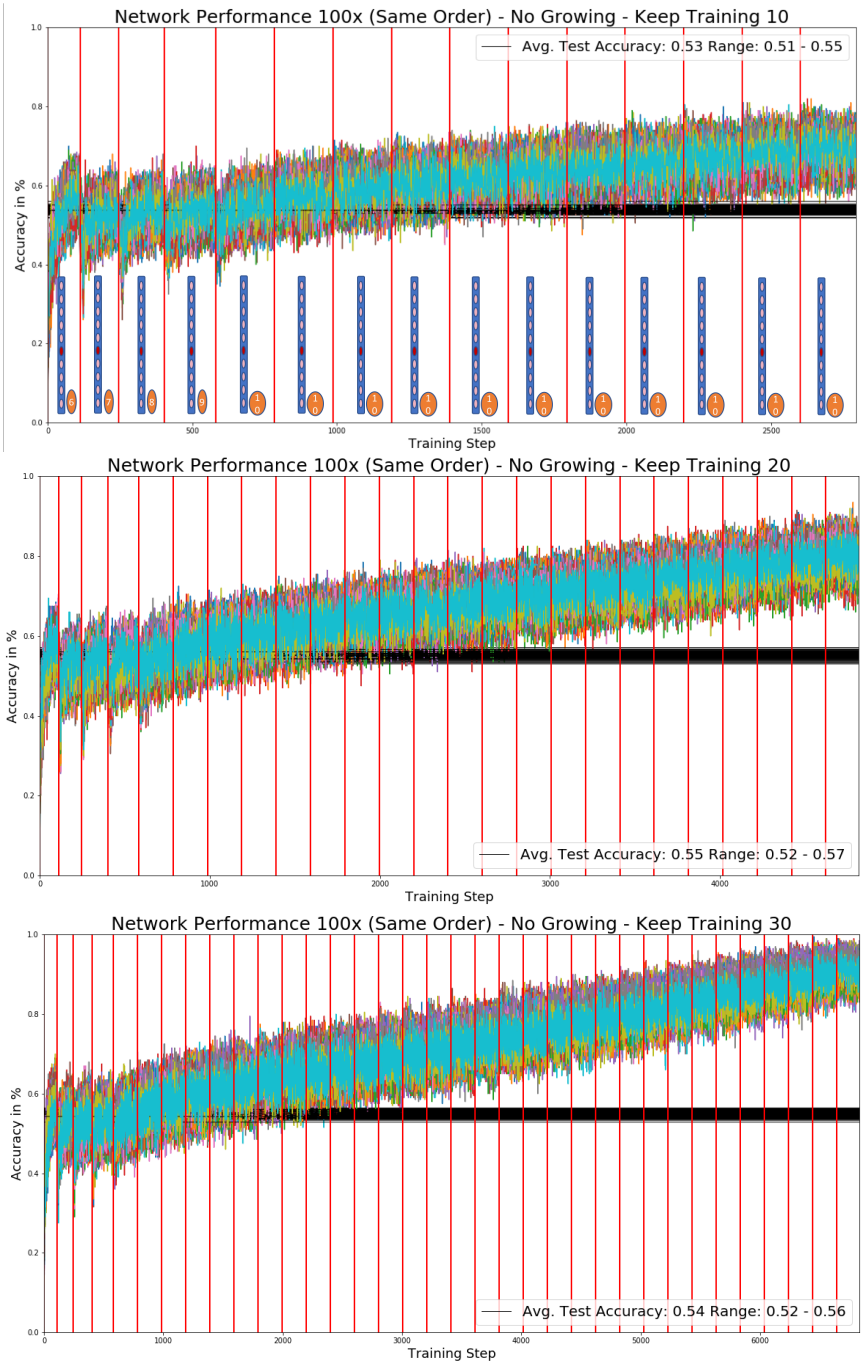

FIGURE A.7: Network accuracy's on CIFAR-10, iterative class introduction, no growing. (top) Train for ten more epochs after the last class has been introduced. (middle) Train for 20 more epochs after the last class has been introduced. (bottom) Train for 30 more epochs after the last class has been introduced.

A.2.6 Growing

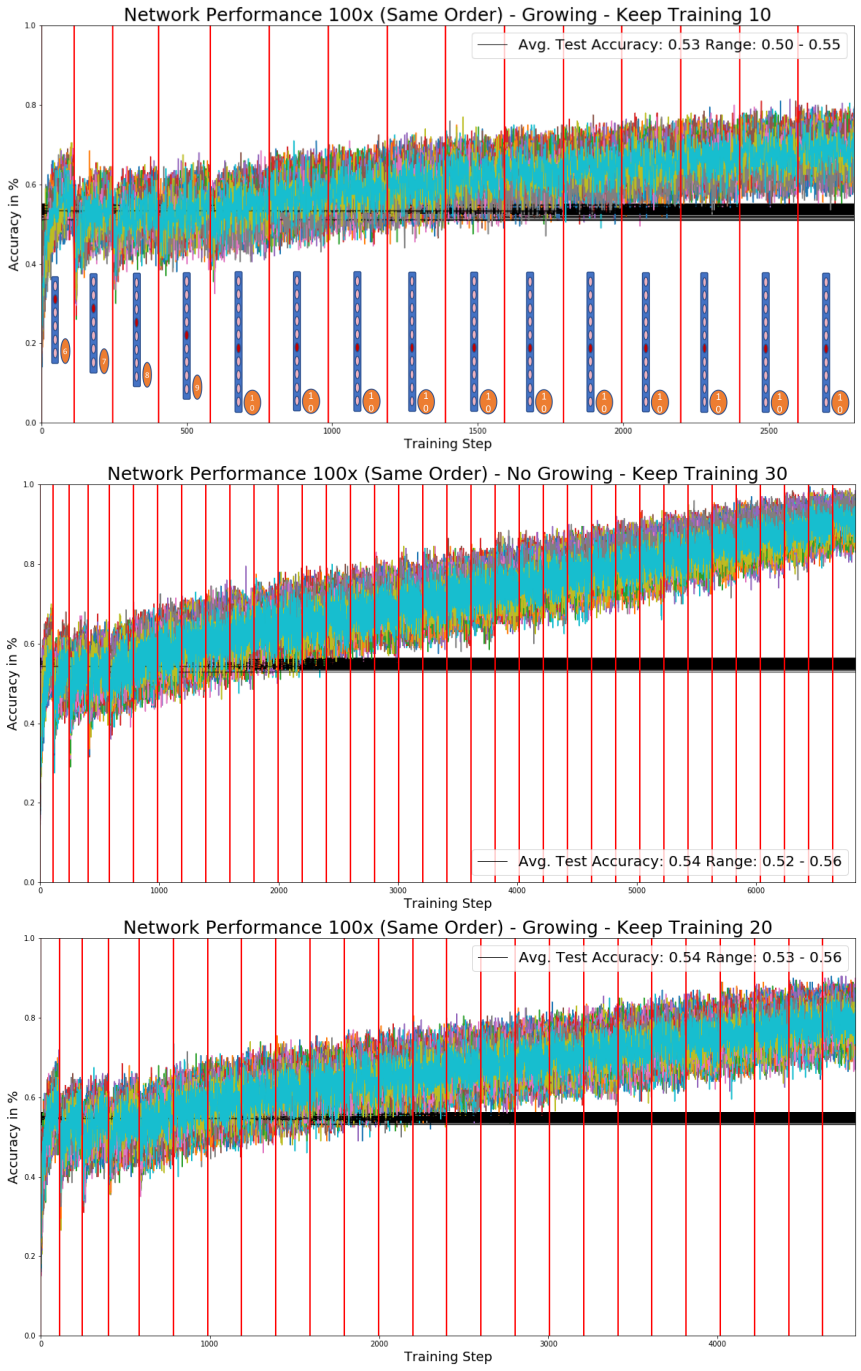

FIGURE A.8: Network accuracy's on CIFAR-10, iterative class introduction, growing last layer. (top) Train for ten more epochs after the last class has been introduced. (middle) Train for 20 more epochs after the last class has been introduced. (bottom) Train for 30 more epochs after the last class has been introduced.

A.2.7 Network Freezing

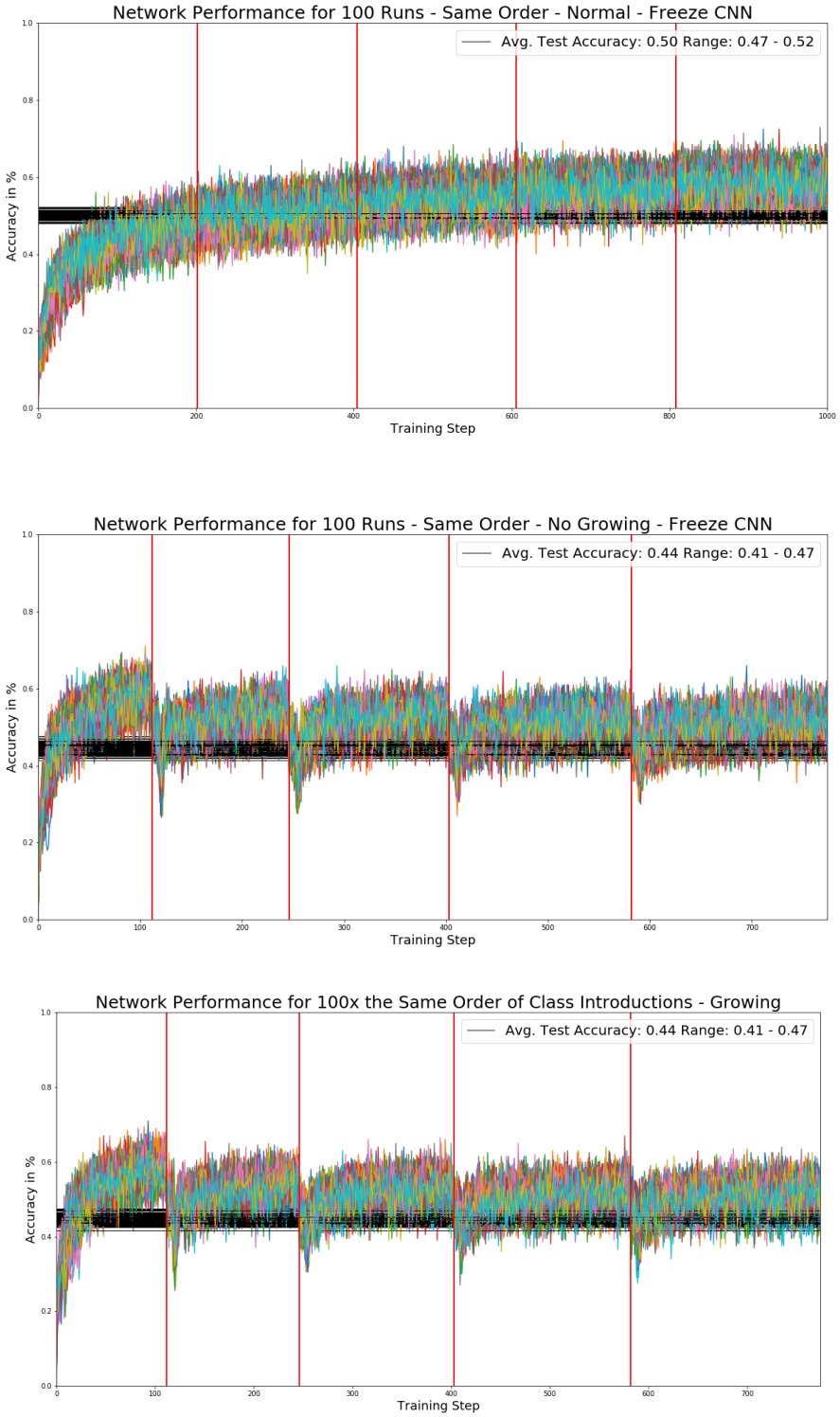

FIGURE A.9: Network accuracy's on CIFAR-10 over 100 runs while freezing the convolutional layer after the first epoch. (Top) Normal training. (Middle) No growing. (Bottom) Growing.

A.3 CIFAR-100 Experiments

A.3.1 Continuous Learning (No Growing) - Results

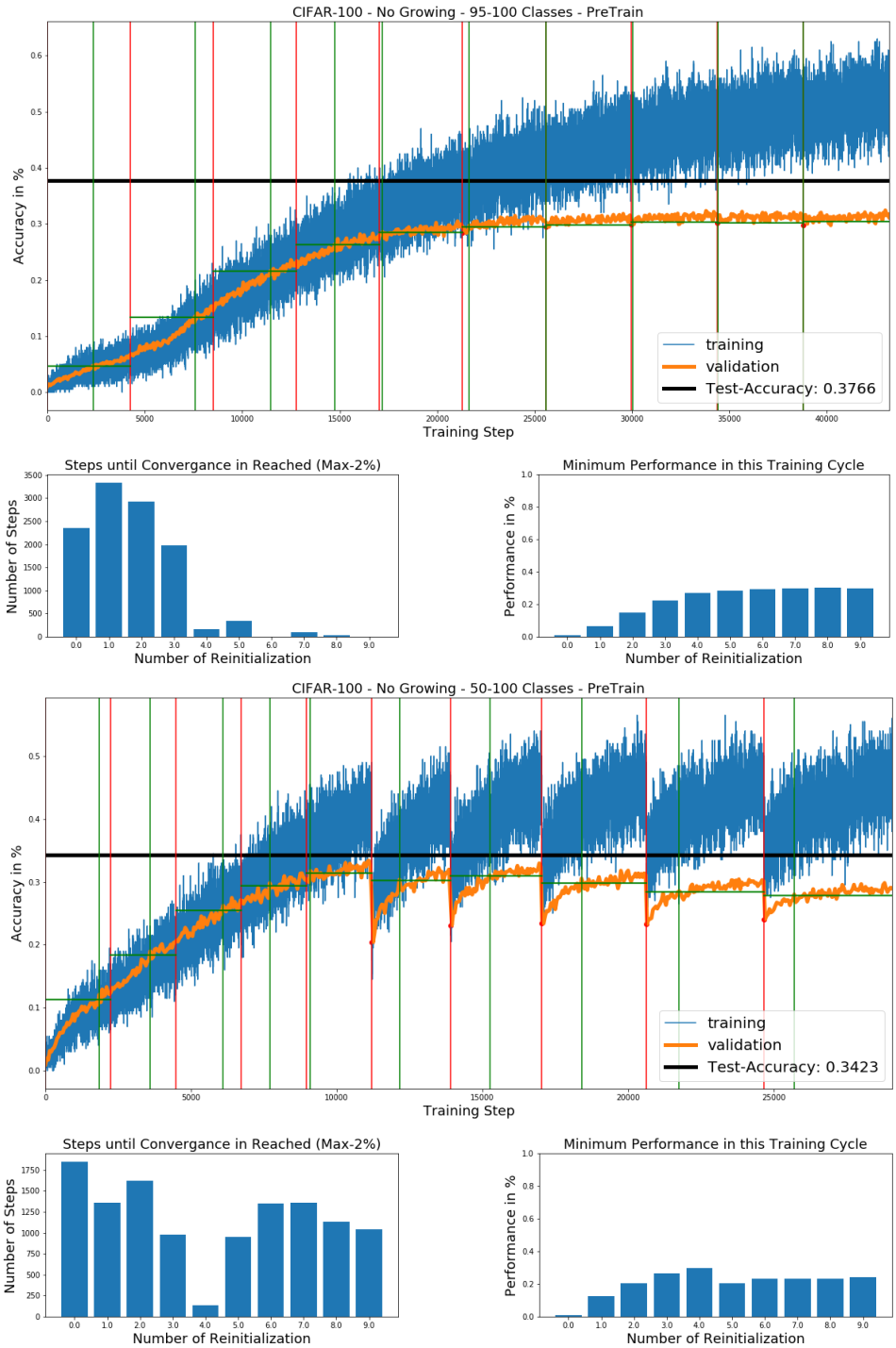

FIGURE A.10: Network accuracy's on CIFAR-100 with continuous class introduction without growing and with five epoch of pre-training. (Top) Introduce one class each epoch. Starting with 95 up to 100 classes. (Bottom) Introduce ten classes each epoch. Starting with 50 classes up to 100.

A.3.2 Weight Changes for Each Node

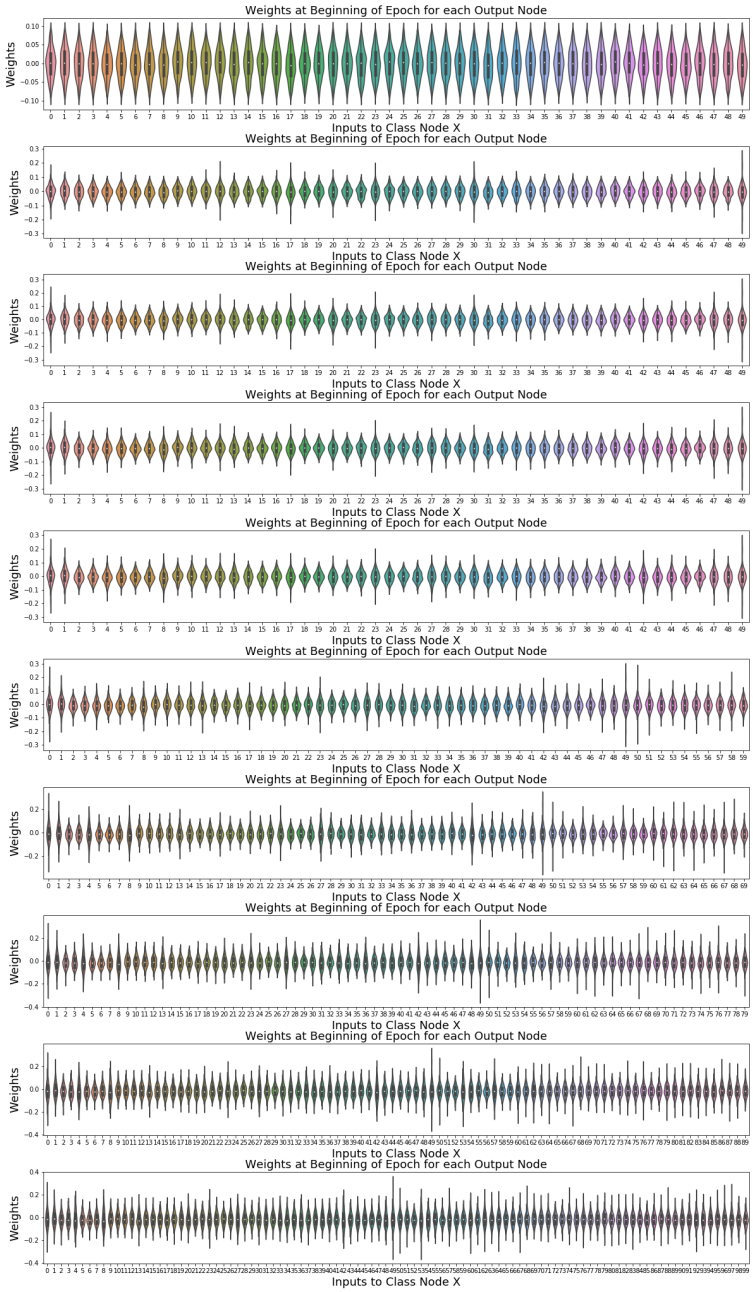

FIGURE A.11: Distribution of weights in the last layer at the beginning of each of ten epochs (from top to bottom). Network is trained on CIFAR-100 with five epochs of pre-training with 50 classes. Starting in epoch 5 ten new classes are introduced in each epoch until in epoch 9 all 100 classes are in the data set. The last layer of the network grows with the number of classes present in the data. Each violin represents the weights connecting to one of the output nodes. Staring from row 5 the leftmost ten violins show the weight connecting to the ten new nodes.

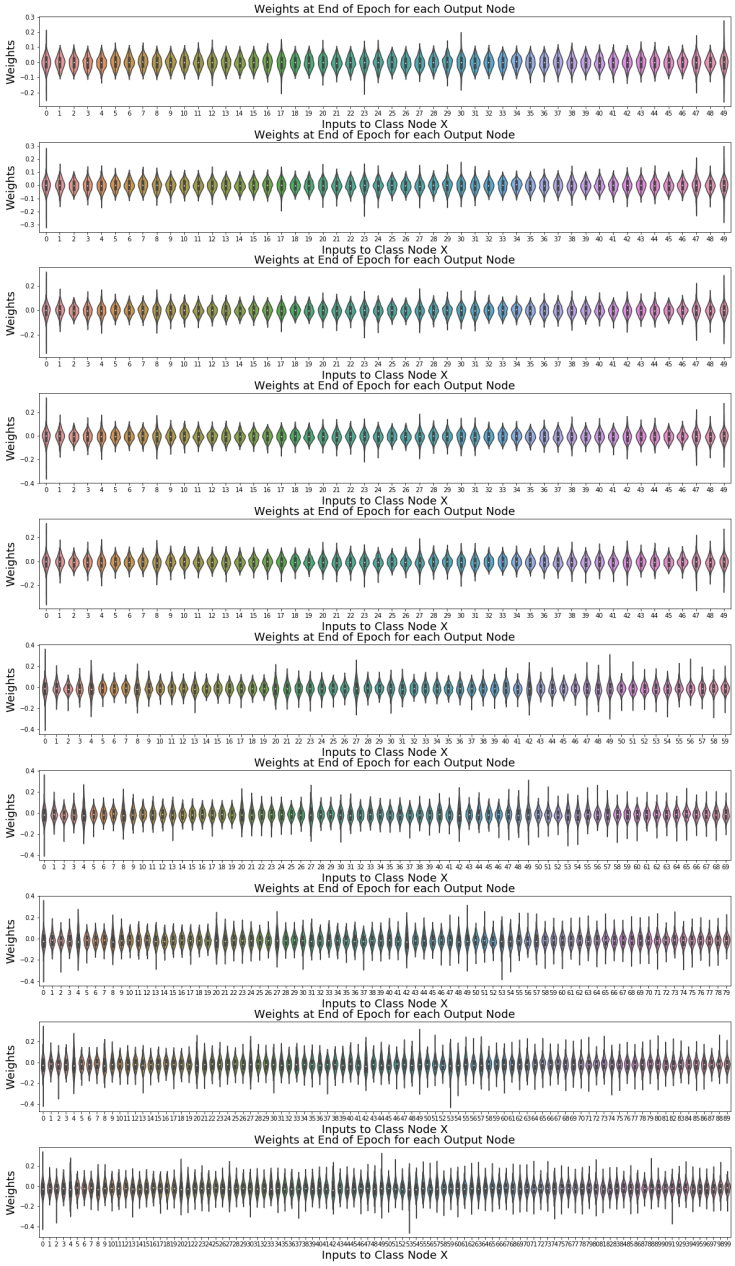

FIGURE A.12: Distribution of weights in the last layer at the end of each of ten epochs (from top to bottom). Network is trained on CIFAR-100 with five epochs of pre-training with 50 classes. Starting in epoch 5 ten new classes are introduced in each epoch until in epoch 9 all 100 classes are in the data set. The last layer of the network grows with the number of classes present in the data. Each violin represents the weights connecting to one of the output nodes. Staring from row 5 the leftmost ten violins show the weight connecting to the ten new nodes.

Bibliography

Abraham, Wickliffe C and Anthony Robins (2005). "Memory retention – the synaptic stability versus plasticity dilemma". In: 28.2. DOI: 10.1016/j.tins.2004.12.003.

Asif, Umar, Mohammed Bennamoun, and Ferdous Sohel (2016). "Unsupervised segmentation of unknown objects in complex environments". In: *Autonomous Robots* 40.5, pp. 805–829. ISSN: 15737527. DOI: 10.1007/s10514-015-9495-3.

Bengio, Yoshua et al. (2009). "Curriculum learning". In: *Proceedings of the 26th annual international conference on machine learning*, pp. 41–48. ISSN: 0022-5193. DOI: 10.1145/1553374.1553380. arXiv: arXiv:1011.1669v3. URL: http://portal.acm.org/citation.cfm?doid=1553374.1553380.

Coop, Robert, Aaron Mishtal, and Itamar Arel (2013). "Ensemble learning in fixed expansion layer networks for mitigating catastrophic forgetting". In: *IEEE Transactions on Neural Networks and Learning Systems* 24.10, pp. 1623–1634. ISSN: 2162237X. DOI: 10.1109/TNNLS.2013.2264952.

Daniels, Zachary A. and Dimitris N. Metaxas (2018). "Scenarios: A New Representation for Complex Scene Understanding". In: arXiv: 1802.06117. URL: http://arxiv.org/abs/1802.06117.

Dvornik, Nikita et al. (2017). "BlitzNet: A Real-Time Deep Network for Scene Understanding". In: *Proceedings of the IEEE International Conference on Computer Vision* 2017-Octob, pp. 4174–4182. ISSN: 15505499. DOI: 10.1109/ICCV.2017.447. arXiv: 1708.02813.

Elman, L (1993). "Learning and development in neural networks : the importance of starting small". In: 48, pp. 71–99.

Gepperth, Alexander and Cem Karaoguz (2016). "A Bio-Inspired Incremental Learning Architecture for Applied Perceptual Problems". In: *Cognitive Computation* 8.5, pp. 924–934. ISSN: 1866-9956. DOI: 10.1007/s12559-016-9389-5. URL: http://link.springer.com/10.1007/s12559-016-9389-5.

Glorot, Xavier, Antoine Bordes, and Yoshua Bengio (2011). "Deep sparse rectifier neural networks". In: *AISTATS '11: Proceedings of the 14th International Conference on Artificial Intelligence and Statistics* 15, pp. 315–323. ISSN: 15324435. DOI: 10.1.1.208.6449. arXiv: 1502.03167.

Graves, Alex et al. (2017). "Automated Curriculum Learning for Neural Networks". In: ISSN: 1938-7228. arXiv: 1704.03003. URL: http://arxiv.org/abs/1704.03003.

Hinton, Geoffrey E. and David C. Plaut (1987). *Using Fast Weights to Deblur Old Memories*.

Ioffe, Sergey and Christian Szegedy (2015). "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: ISSN: 0717-6163. DOI: 10.1007/s13398-014-0173-7.2. arXiv: 1502.03167. URL: http://arxiv.org/abs/1502.03167.

Johnson, Justin, Andrej Karpathy, and Li Fei-Fei (2015). "DenseCap: Fully Convolutional Localization Networks for Dense Captioning". In: ISSN: 10636919. DOI:

10.1109/CVPR.2016.494. arXiv: 1511.07571. URL: http://arxiv.org/abs/1511.07571.

Karpathy, Andrej (2016). "Connecting Images and Natural Languag". In: August, p. 95.

Karpathy, Andrej, Armand Joulin, and Li Fei-Fei (2014). "Deep Fragment Embeddings for Bidirectional Image Sentence Mapping". In: pp. 1–9. ISSN: 10495258. arXiv: 1406.5679. URL: http://arxiv.org/abs/1406.5679.

Kemker, Ronald et al. (2017). "Measuring Catastrophic Forgetting in Neural Networks". In: ISSN: 1091-6490. DOI: 10.1073/pnas.1611835114. arXiv: 1708.02072. URL: http://arxiv.org/abs/1708.02072.

Kingma, Diederik P. and Jimmy Ba (2014). "Adam: A Method for Stochastic Optimization". In: pp. 1–15. ISSN: 09252312. DOI: http://doi.acm.org.ezproxy.lib.ucf.edu/10.1145/1830483.1830503. arXiv: 1412.6980. URL: http://arxiv.org/abs/1412.6980.

Kirkpatrick, James et al. (2017). "Overcoming catastrophic forgetting in neural networks". In: 114.13, pp. 3521–3526. DOI: 10.1073/pnas.1611835114.

Krushke, John K. (1992). "ALCOVE: An Exemplar-Based Connectionist Model of Category Learning". In: *Psychological Review* Vol.99.No. 1, pp. 22–44. URL: http://cognitrn.psych.indiana.edu/rgoldsto/courses/concepts/Kruschke1992.pdf.

Li, Li Jia, Richard Socher, and Li Fei-Fei (2009). "Towards total scene understanding: Classification, annotation and segmentation in an automatic framework". In: *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops, CVPR Workshops 2009*, pp. 2036–2043. ISSN: 1063-6919. DOI: 10.1109/CVPRW.2009.5206718.

Murre, Jacob M.J., R. Hans Phaf, and Gezinus Wolters (1992). "CALM: Categorizing and learning module". In: *Neural Networks* 5.1, pp. 55–82. ISSN: 08936080. DOI: 10.1016/S0893-6080(05)80007-3. URL: http://linkinghub.elsevier.com/retrieve/pii/S0893608005800073.

Polikar, Robi et al. (2001). "Learn++: An incremental learning algorithm for supervised neural networks". In: *IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews* 31.4, pp. 497–508. ISSN: 10946977. DOI: 10.1109/5326.983933.

Ratcliff, Roger (1990). "Connectionist Models of Recognition Memory: Constraints Imposed by Learning and Forgetting Functions". In: *Psychological Review* 97.2, pp. 285–308. ISSN: 0033295X. DOI: 10.1037/0033-295X.97.2.285.

Robins, Anthony (1995). "Catastrophic Forgetting, Rehearsal and Pseudorehearsal". In: *Connection Science* 7.2, pp. 123–146. ISSN: 13600494. DOI: 10.1080/09540099550039318.

Russakovsky, Olga, Li Jia Li, and Li Fei-Fei (2015). "Best of both worlds: Human-machine collaboration for object annotation". In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* 07-12-June, pp. 2121–2131. ISSN: 10636919. DOI: 10.1109/CVPR.2015.7298824.

Scherer, Dominik, Andreas Müller, and Sven Behnke (2010). "Evaluation of pooling operations in convolutional architectures for object recognition". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 6354 LNCS.PART 3, pp. 92–101. ISSN: 03029743. DOI: 10.1007/978-3-642-15825-4_10.

Teng, Ervin, Rui Huang, and Bob Iannucci (2018). "ClickBAIT-v2: Training an Object Detector in Real-Time". In: arXiv: 1803.10358. URL: https://arxiv.org/pdf/1803.10358.pdf.

Venkatesan, Rajasekar and Meng Joo Er (2016). "A novel progressive learning technique for multi-class classification". In: *Neurocomputing* 207, pp. 310–321. ISSN: 18728286. DOI: 10.1016/j.neucom.2016.05.006. arXiv: 1609.00085.

Viola, Paul and Michael J Jones (2001). "Robust Real-time Object Detection". In: *International Journal of Computer Vision*. ISSN: 09205691. DOI: 10.1.1.23.2751.

Xu, Tao et al. (2017). "AttnGAN: Fine-Grained Text to Image Generation with Attentional Generative Adversarial Networks". In: arXiv: 1711.10485. URL: http://arxiv.org/abs/1711.10485.

Zhang, Han et al. (2017). "StackGAN: Text to Photo-Realistic Image Synthesis with Stacked Generative Adversarial Networks". In: *Proceedings of the IEEE International Conference on Computer Vision* 2017-Octob, pp. 5908–5916. ISSN: 15505499. DOI: 10.1109/ICCV.2017.629. arXiv: 1612.03242.

Zhu, Yuke et al. (2015). "Building a Large-scale Multimodal Knowledge Base System for Answering Visual Queries". In: arXiv: 1507.05670. URL: http://arxiv.org/abs/1507.05670.