Aalborg University Copenhagen

Semester: MED10

Title:
Exploring gradual learning rates for transfer learning in an image classification setting

Project Period:
Spring 2018 (Feb 2018 - May 2018)

Semester Theme:
Master Thesis

Supervisor(s):
Hendrik Purwins
(hpu@create.aau.dk)

Project group no.:
None given.

Members:
Esben Winther Knudsen
Study nr.: 20136320
(ewkn13@student.aau.dk)

Sebastian Siem Bach-Nielsen
Study nr.: 20136819
(sbachn13@student.aau.dk)

Copies: 1
Pages: 55
Finished: 31/05-2018

Aalborg University Copenhagen

Frederikskaj 12,

DK-2450 Copenhagen SV

Semester Coordinator: Stefania Serafin

Secretary: Lisbeth Nykjær

Abstract:

This project investigates the combined effectiveness of state-of-the-art transfer learning methods, here including Learning Rate Finder, Warm Restarts, Gradual Learning Rates and Cosine Annealing. These methods was recently proposed by Howard & Ruder (2018) but empirical experiments has never been attempted, in a supervised image classification context.

It also analyses the Modified Hausdorff Distance measure (MHD) as a possible estimator for deciding the weight hyperparameter for gradual learning rates. The dissimilarity measure seemed to make intuitive sense when estimating the marginal distribution of the two datasets. Yet showed no noteworthy correlation between the performance increase of the target-set and the dissimilarity distance to the source-set. Further analysis of the experimental results however, opened up a future prospect on analysing how the combined class variance might be a more promising feature to use as a measure for predicting knowledge transferability between two tasks.

Exploring gradual learning rates for transfer learning in an image classification setting

Authors:
Esben W. Knudsen & Sebastian S. Bach-Nielsen
(ewkn13, sbachn13@student.aau.dk)
Supervisor:
Dr. Hendrik Purwins

Department of Architecture, Design & Media Technology
Aalborg University, Copenhagen

Contents

1 Introduction

In recent years, deep neural networks has become increasingly good at learning a very accurate mapping of high dimensional data inputs, such as images, onto some label prediction. The challenge currently at hand, is the models ability or lack thereof to generalize on input transformations that are different from the once encountered during training. Generalizing on a task is considered *the* task within the overall field of artificial intelligence, as it essentially represent robustness of a trained model's ability to accurately predict on the infinite number of novel input scenarios, the real world actually consist of. One of the most promising answers in the recent years, within the study in text and image classification, is called transfer learning. The method refers to the ability to transfer valuable knowledge obtained from one task onto another, and will be thoroughly discussed throughout this report. This method is considered one of the biggest current drivers for machine learning in the commercial world (Ng, 2016). Several of the multi million dollar companies in the tech world such as google, actually offers a large variety of free machine learning models, already trained to perform very general tasks, on extremely large datasets ("Word2vec", n.d.; "Pre-trained models" n.d.). The overall idea is then to finetune one of those pre-trained models to your own task at hand. However fine-tuning the model with the right amount is a delicate matter, as too much re-training can cause catastrophic forgetting, thereby losing otherwise valuable knowledge from the previous task (Howard & Ruder, 2018). Near the end of this report several empirical tests on some of the state-of-the-art approaches when performing transfer learning will be investigated. In addition, the report aims to search for a possible similarity or dissimilarity measure for estimating how much relevant information can be transferred between to two tasks. Yet, before digging into the implementation of these two objectives, a more clear understand of what transfer learning is must be established.

2 Definition of Transfer Learning

This study calls for a deep understanding of what transfer learning is and how it has been used. This section will therefore focus on defining the term within machine learning, while investigating different implementations that are considered most successful. Transfer learning refers to the method where information and knowledge gained from pre-training a model towards a certain task, often denoted as a source task, can be transferred and used to create a high performance model for a new target task, if the source and target tasks are considerably related (Weiss et al, 2016).

The general outcome of training a machine learning model on a dataset, is to tune its parameters such that it can fit a particular input (e.g. en image) to some output (a label). The amount of parameters needed is proportional to the complexity of the task the model has to perform. However, as the parameters of the model increases, so does the amount of data needed for tuning them. The need for transfer learning occurs mainly when there are scarce amount of data available, making the creation of a well performing model a difficult task. The scarcity could be due to data being rare, inaccessible, expensive and/or time consuming to obtain, which is quite common in a real world scenario. Fortunately, as big data repositories becomes more extensive and publically available, e.g. Kaggle ("Welcome to Kaggle Datasets", n.d.), ImageNet (Russakovsky et al., 2015), Cifar10 (Krizhevsky, 2009), finding existing data sets that are similar to your own task, is often possible. This makes transfer learning a feasible and appealing machine learning approach (Weiss et al, 2016).

Finding ways to alternatively acquire more data in order to increase a models ability to generalize is not an unusual phenomenon. Another very common trick closely related is called data augmentation. It transforms your current data into multiple slightly altered instances, thereby increasing the amount of data points available for training. In the case of a image-dataset, the input can be changed with e.g. rotating, cropping or noise inducting the original image, without altering the overall semantic meaning of the given picture (its label). Whether it is augmenting on the data that you currently have, or finding a similar dataset to train you model on first, it's all with the motivation of increasing the amount of data your model sees to make it invariant to a wide variety of transformations (Goodfellow et al., 2016, p. 236).

Pan et al. (2010) and Weiss et al. (2016) have both done an extensive study on the field of transfer learning that we will closely follow, while interpreting it all for a image classification problem. They both define transfer learning as concerning the concepts of a domain $\mathcal{D}$ and a task $\mathcal{T}$. The domain $\mathcal{D}$ consists of two components, namely, a feature space $\mathcal{X}$ and a marginal probability distribution P(X) over the feature space, where X $= \{x_1, ..., x_n\} \ \epsilon \ \mathcal{X}$. In regard to an image classification problem, $\mathcal{X}$ is the feature space where all images are located, x_i is equal to the ith image instance and X is the sample of images used for training.
Given a domain $\mathcal{D} = \mathcal{X}, P(X)$, a task consists of two components; A label space $\mathcal{Y}$ and a predictive function $f(\cdot)$. The predictive function is tuning through the training data which in turn consisting of pairs $x_i \ \epsilon \ X$ and $y_i \ \epsilon \ \mathcal{Y}$. This classification function is used when predicting the label y_i of its associated x_i and is denoted as the conditional probability distribution $P(Y|X)$. We can therefore write the task as $\mathcal{T} = \mathcal{Y}, P(Y|X)$.
Given a source domain $\mathcal{D}_s$, a corresponding source task $\mathcal{T}_s$, a target domain $\mathcal{D}_t$ and a target task $\mathcal{T}_t$, the objective of transfer learning is to help improve the learning of the target task $\mathcal{T}_t$ using the information gained from training towards the source task $\mathcal{T}_s$ and domain $\mathcal{D}_s$, where $\mathcal{D}_s \neq \mathcal{D}_t \ \ or \ \ \mathcal{T}_s \neq \mathcal{T}_t$ (Pan et al, 2010).

The common method to improve generalization across different distributions between source and target domains and/or tasks is to assume that the feature space between source and target are the same i.e $\mathcal{D}_s = \mathcal{D}_t, or \mathcal{T}_s = \mathcal{T}_t$. This method is also known as homogeneous transfer learning (Weiss et al, 2016). In order to deal with homogeneous transfer learning problems there are three common approaches; (1) Trying to correct for the marginal distribution differences ($P(X)$), (2) trying to correct for the conditional distribution differences ($P(Y|X)$) or (3) trying to correct for both marginal and conditional distribution simultaneously (Weiss et al, 2016).

However, in many real world applications, having the same feature space between source and target is not always given, thus finding a method for transferring knowledge learned from similar tasks, is highly sought for. This method is known as heterogeneous transfer learning and pertains to the differences between source and target domains. Due to the definition of transfer learning four scenarios arise where the conditions may vary, in the sense that real world transfer learning problems might involve a mixture of the conditions. These scenarios are depicted below with examples

pertaining to our image classification problem:

1. $\mathcal{X}_s \neq \mathcal{X}_t$. The feature space of the source and target domains are incomparable. An example could either be the feature difference in images of cats and airplanes - However, Yosinski et al (2014) argues that such two (and most other) image classes may have more in common than initially thought as they share the space of low level features such as edges and color blobs.

2. $P(X_s) \neq P(X_t)$. The marginal probability distributions of source and target domain are inconsistent. An example would be where X_s solely consist of images of elephants and X_t of mice, and they are both labeled 'animal'. Both datasets are part of the animal distribution yet each individual class distributions are not overlapping each other in the high dimensional space.

3. $\mathcal{Y}_s \neq \mathcal{Y}_t$. The label space between the source and target task are different. Here an example could be two datasets of identical images of cats and dogs, however in one of the datasets the gender is labeled, where in the other the specie is labeled - identical data points yet the task is different.

4. $P(Y_s|X_s) \neq P(Y_t|X_t)$. The conditional probability distributions between the source and target tasks are different from each other. An example could be two unbalanced datasets of images of cats and dogs, here the probability of predicting cat or dog differs depending on the dataset the model has been trained on.

2.1 Negative transfer

A substantial majority of all real world datasets that are considered similar, differs with different extends on each of the four conditions. For transfer learning to work properly and indeed enhance the target learner, there must be a close relation in the four conditions stated above, between the source and target at hand (Pan et al, 2010; Weiss et al 2016). But what will happen if this relation is vague or almost non-existing? In such cases, the source learner might induce a negative effect on the target task, causing an unwanted decrease in performance.

The most accepted approach when creating any convolutional neural network, and many other machine learning models, is to initialize its weights with random values either normally or uniformly distributed (*"Variables:*

Creation, Initialization, ...”, 2016; *“Usage of initializers”*, n.d.; *“Source code for torch.nn.init”*, n.d.). Thus, for transfer learning not to have a negative effect, the re-trained model having pre-trained weights, must outperform on the target task, compared to starting with random weights. In literature the effect of obtaining a worse outcome using transfer learning is referred to as ‘negative transfer’, and is unfortunately a seldom researched area (Pan et al, 2010; Weiss et al, 2016). Foreseeing the performance outcome of pre-training a model on similar versus dissimilar datasets has been proven very difficult. The transfer learning method therefore calls for a way to estimate this outcome. When again considering the four conditions presented by Pan et al. (2010) and Weiss et al. (2016), it might be possible to construct separate measures for each condition to somehow make a well-founded assumption on how well knowledge from one dataset can be transferred onto another.

One condition that might be feasible to investigate is the 2nd. It suggests that dataset distributions that lies close to each other in high-dimensional-space, may share some similarity. In this report we investigate how well the Modified Hausdorff distance measure (Dubuisson & Jain, 1994) can be used for such purpose. This distance measure will be discussed further later in the report.

However, It could be argued that some for the conditions it might be impossible to systematically evaluate, or even redundant to do so; For instance, estimating the feature space of the source and target domains (condition 1) calls for some form of low to high level feature extraction. The current state-of-the-art method for extracting relevant task features are through a machine learning model trained on the specified dataset. However, if you have a model that can already extract those genuinely true target features, pre-learning on a source dataset becomes completely redundant, since the true features should be able to, by definition, separate your target classes perfectly. Therefore, using a model to extract features of two datasets and after somehow measure the similarity between the two feature sets, with the purpose of estimating how well pre-training on the source dataset can increase the target task, seem rather counterintuitive.

The next section will aim to establish a better understanding of the different studies done in the field of knowledge transferability from one neural network to another. Two papers in particular will be covered; *"How transferable are features in deep neural networks"* by Yosinski et al (2014), who

makes a large empirical study on the transferability of different layers and when layers go from general to task specific, and another more recent paper by Howard & Ruder (2018) who introduces a new method of re-training a model, called 'Discriminative Finetuning' with 'Gradual Learning Rates'.

3 Related works

3.1 From general to specific

Yosinski et al (2014) investigates when features in a deep neural network transitions from general to specific throughout the different layers as well as how this transition affects the transferability. In Parallel different transfer learning techniques, such as randomly initializing weights, freezing layers and fine-tuning are tested to see their influence on the performance. The basis of their research spawn from the intuitive notion that the first layers of a deep neural network finds small general features, such as Gabor filters and color blobs, whereas the last layers of the network finds features that are more specific towards the given task (Yosinski et al, 2014). They define a set of features learned from some task A (source), are general if it can be used for another task B (target). Furthermore, they argue that layer generality is dependant on the similarity between the datasets used for task A and B, reaffirming the 2nd condition given by Pan et al (2010) and Weiss et al (2016).

They run two experiments using the currently largest dataset available called 'ImageNet' consisting of 1000 classes with a total of about 1.4 million images. In the first experiment they test which layer the network transitions from general to task specific. They divide ImageNet into two datasets each consisting of 500 randomly chosen classes. Then two convolutional neural networks similar to the Alexnet architecture (5 conv. and 3 fully connected layers) are trained, resulting in a network for each dataset. The networks are called baseA and baseB. From the base models two new networks are made per layer (Yosinski et al, 2014);

1. A *control network* where the first n amount of layers (e.g. 1-3) from baseA are copied and frozen. Then the next n amount of higher layers (e.g. 4-8) are initialized by random and trained on dataset A. By doing so a baseline for the performance is found.

2. A*transfer network* where the first n amount of layers from baseB are copied and frozen. Then the next$n$ amount of higher layers are initialized randomly towards dataset B.

If the *transfer network* (2) performs as well as the *control network* (1) it shows that n-th layer features are general, towards task B. If performance drops then the n-th layer features are specific towards task A. Yosinski et al.

(2014) repeats this process for all number of layers *(1,2,...,8)*, and likewise reiterate the process using fine-tuning instead of freezing the layers.

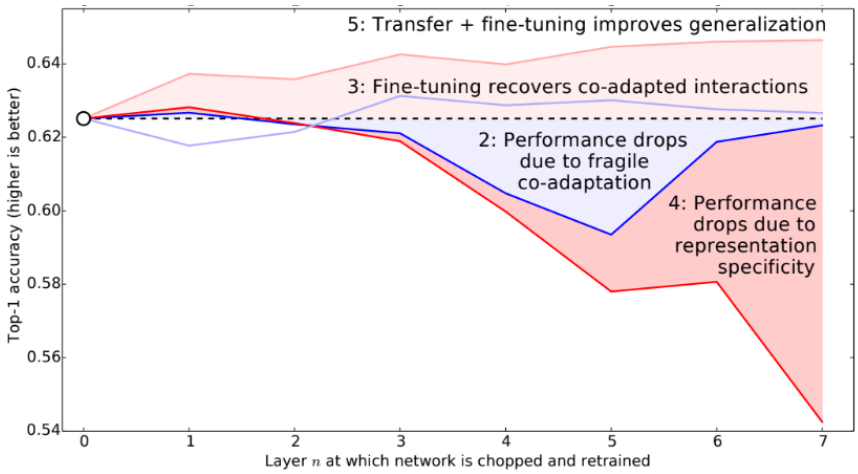


Figure 1: *Lines connecting the means of each treatment. Dark blue line represents a BnB network. Light blue line represent BnB+ networks, or fine-tuned versions of BnB. Dark red line is AnB networks, and light red lines are the fine-tuned AnB+ versions (Yosinski et al., 2014, p. 5).*

The experimental results using layer-wise freezing does indeed show that the later layers of the network are less general, as performance drop notably when transferring features from the last layers. The results regarding the effectiveness of fine-tuning shows very promising results. It increases the performance throughout all layers compared to both the base performances as well as having layers frozen. In particular fine-tuning can help recover the *fragile co-adaptations* observed on the control network and improves overall generalization (see figure 1). Indicating that fine-tuning all layers is the superior technique, a notion that also reported by Howard et al (2018) and Girshick et al (2014). Furthermore, a surprising behaviour is found in the *control network*, where a performance drop occurs in the middle layers. This is argued to be due to *fragile co-adaptation*, where the successive layers in the network cannot recreate the same co-adapted features as in the *base-network*. No further elaboration is given and whether it may be due to the structure of the network needs additional investigation as it is merely suggested that it occurs in the convolutional layers, and not in the fully connected ones.

The second experiment performed differ only in the way that two new datasets are created that are supposed to represent two distant tasks that are as semantically different as possible. This is achieved by Yosinski et

al. (2014) manually dividing the ImageNet into two distinct datasets, one being images of only man-made objects (e.g. buildings, tools, cars, etc.) while the other dataset only contains images of natural objects (e.g. trees, rivers, animals, etc.). Once again the above process is iterated using the dissimilar datasets.

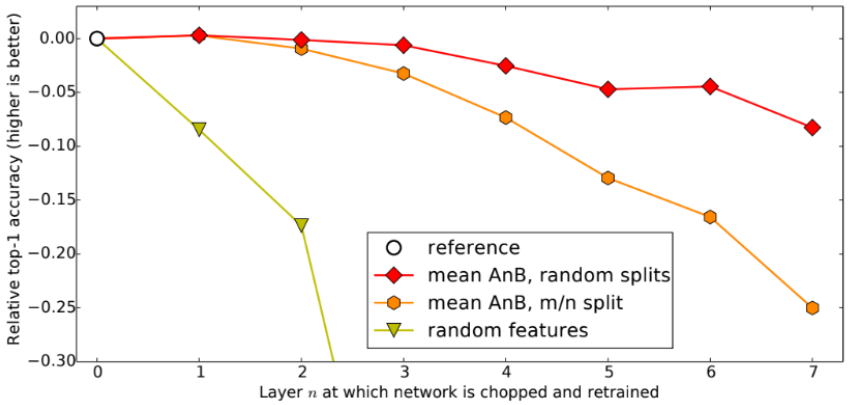


Figure 2: *Performance degradation vs. layer. Comparing the performance plot of networks trained on of the random A/B split (red diamonds) and "natural vs man-made" A/B split (orange hexagons), all normalized by subtracting their base level performance (Yosinski et al., 2014, p. 8).*

When analyzing the results of the dissimilar datasets it can be seen that the performance drops compared to datasets that are, to some extent, similar (see figure 2). This corresponds with the various definitions mentioned earlier, that the task must be related in some way. Unfortunately, the extent to which the two datasets are different is not precisely quantified in Yosinski et al.'s experiment - Other than stating one containing images of "man-made" objects and the other of "natural" objects. In other words, the authors give no objective or standardize measure for describing the relatedness between its two datasets.

Similar to the work by Yosinksi et al (2014), Azizpour et al (2015) also investigates the generic and specific features produced a by convolutional neural network. Their research revolves around finding what factors of a network that can be optimized for various transfer learning tasks and how transferability is affected by the distance between these tasks. The order in which the tasks are reported is equivalent to how distant they are supposed to be for one another. The list is as follow; Image Classification, Attribute Detection, Fine-grained Recognition, Compositional and Instance Retrieval (Azizpour et al., 2015). Thus, the tasks furthest apart is the image classi-

fication task and the instance retrieval task.

What Azizpour et al. (2015) discovered from the experiment is that factors like dimensionality reduction, using techniques such as principal component analysis (PCA), can prove helpful for reducing the curse of dimensionality as well as boost performance for more distant tasks. Furthermore, they find that fine-tuning has a positive impact on performance even when the distance between the datasets are large, which is coherent with the findings of Yosinski et al (2014). However, similar to Yosinski et al (2014), no quantified distance measure is reported upon, in order to find the distance between the tasks referred to throughout the paper. The reasoning and arguments for the distance between the tasks are purely based on the authors own semantic consideration of the similarity between said tasks.

3.2 Discriminative Finetuning

In the very recent paper on "Discriminative Fine-tuning" written by Howard & Ruder (2018) a set of renewed methods, that can be applied when performing fine-tuning for language or image classification tasks, are introduced. Sequentially, they state that fine-tuning the target learner is one of the most crucial parts when transferring knowledge, and should be carefully executed. If one is too overly aggressive when fine-tuning, it can result in fatal forgetting and eradicate the useful information gained from training on the source task. Vice versa, if one is too cautious when performing fine-tuning the convergence speed will be extremely low, possible resulting in overfitting on the target task.

Neither of these two scenarios are desired, and earlier solutions have dealt with this problem by not fine-tuning on all layers at once, but rather one layer at a time. This method is known as *Chain-Thawing* (Felbo et al., 2017; Howard & Ruder, 2018). The chain-thaw approach is implemented by sequentially unfreezing and fine-tuning a single layer at a time. Once each layer of the network have been unfrozen and fine-tuned all layers are fine-tuned collectively an additional time. This method increases the accuracy on the target task, however, with the expense of more computational power needed for fine-tuning (Felbo et al., 2017). Howard & Ruder (2018) have experimented with a similar technique to that of Felbo et al (2017) referred to as *Gradual Unfreezing*. Instead of unfreezing one layer, fine-tune it, freeze it again, and then repeat the process for each layer at a time, gradual unfreezing is carried out by unfreezing the last hidden layer and

then fine-tune. Subsequently a new layer is unfrozen and the two unfrozen layers are then fine-tuned collectively, etc. This process is then iterated until the entire network has been fine-tuned.

Even though chain-thawing or gradually unfreezing are viable methods for transfer learning they introduce a sequential requirement that may result in overfitting, as the target dataset is elapsed each time a layer is unfrozen and fine-tuned (Howard & Ruder, 2018). This pose a substantial drawback as datasets used for transfer learning are often small due to limited availability or scarcity, thus being even more prone to overfitting. To overcome this limitation Howard & Ruder (2018) suggest to perform *'discriminative fine-tuning'*, a technique proposed in their paper. The main concept of discriminative fine-tuning is that different layers of a network attain different features, thus it makes sense that layers should be fine-tuned to different extents. By fine-tuning earlier layers of a network (those that are general) to a lesser extent than later layers (the specific ones), the previous knowledge can be maintained and used to improve the performance while training towards the target task (Howard & Ruder, 2018). Discriminative fine-tuning is a method that allows one to change the learning rate for each layer individually instead of using a single learning rate for all layers. For clarification Howard & Ruder (2018) use stochastic gradient descent (SGD) to explain how one can split and update the learning rate parameters in order to implement discriminative fine-tuning. For SGD the model's parameters θ, are updated at a timestep t (Ruder, 2016; Howard & Ruder, 2018), as follows:

$$\theta_t = \theta_{t-1} - \eta \cdot \nabla_\theta J(\theta) \tag{1}$$

Here η is the learning rate while $\nabla_\theta J(\theta)$ is the gradient in regard to the models predictive function. The model's parameter θ can be split into $\{\theta^l, ..., \theta^L\}$ where θ contains all the model parameters at the l-th layer while L is the total number of layers in the model. Similarly, the learning rate is obtained for $\{\eta^l, ..., \eta^L\}$ where η^l is equal to the learning rate at the *-th layer* (Howard et al, 2018).

In regard to discriminative fine-tuning, the SGD update will be as follows:

$$\theta_t^l = \theta_t^l - \eta^l \cdot \nabla_{\theta_t} J(\theta) \tag{2}$$

Howard & Ruder (2018) empirically found it to work well to choose a specific learning rate for the last layer, and during fine-tuning use $\eta^l =$

$\eta^{l+1} \cdot 0.3$ for earlier layers, thus reducing the learning rate by 70% per layer. However, a shortcoming of the experiment is that the empirical learning rate decrease seems specific for the task at hand, and is displayed in regard to a natural language processing (NLP) problem. Thus it raises a set of question in regards to its use; (1) is the effect of using layer-wise fine-tuning useful for other transfer learning tasks than NLP, such as in image classification, and (2) whether the reported decrease of 70% per layer is a notion that works for tasks in general or is it specific to the task presented in the paper.

3.2.1 Cosine annealing and warm restarts

In addition to discriminative fine-tuning, Howard & Ruder (2018) mentions another trick that they found useful for their experiments. This trick is based on a method known as cyclical learning rate, where the idea is to allow the learning rate to vary within a range of two values, a minimum and maximum boundary, instead of having the more traditional fixed or stepwise decreasing learning rate (Smith, 2017). An example of a cyclical learning rate schedule can be seen on figure 3.

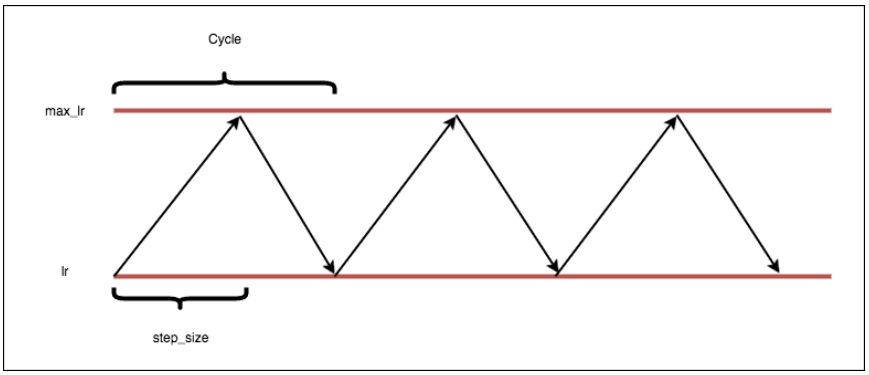


Figure 3: *An example of a cyclical learning rate*

The intuition of using a cyclical learning rates comes from the devious tasks of minimizing the loss during training and getting stuck in either a poor local minima or a flat saddle point. As the gradient is very small for saddle points the learning rate decreases, ultimately slowing down the convergence process. However, by boosting the learning rate every now and then, one is able to move beyond the saddle point as well as a poor minima, resulting in faster convergence and optimally ending in good minima (Smith, 2017).

As cyclical learning rates have shown useful and the results thereof increases performance and convergence speed, practitioners have been experimenting with different types of cyclical learning rates. One method that has been reported to be well functioning is the aggressive cosine annealing (Loshchilov and Hutter, 2016; Howard & Ruder, 2018). By using half a cosine as the annealing schedule one gets a schedule that has a high learning rate to begin with and then smoothly slows down over the iterations. Thus the same use of an upper and lower bound is used, removing the need to manually selecting the decrease of the learning rate. Please refer to figure 4 to see an example of the cosine annealing learning rate schedule.

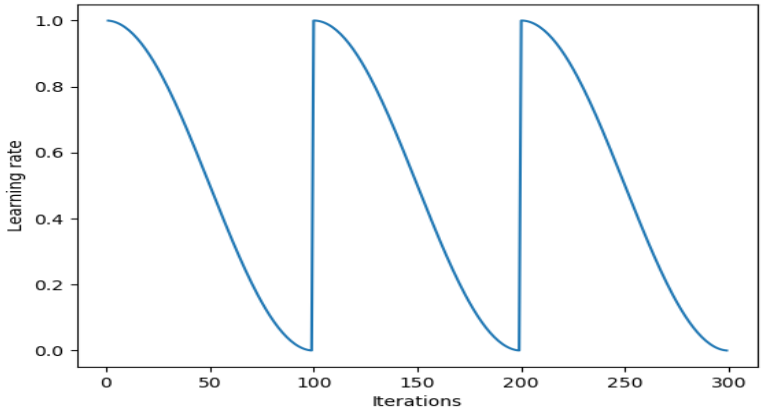


Figure 4: *An example of a cyclical learning rate using cosine annealing*

Furthermore, by applying a reversed cosine annealing schedule for one epoch on the dataset, have empirically shown a performance boost (Smith and Topin, 2017). This method is also known as a warm-up. This means that for one iteration over the dataset, instead of lowering the learning rate, it is gradually increasing. The result thereof removes some of the noise that can be experience due to batch sizes and help with generalization (Smith and Topin, 2017).

3.3 The Modified Hausdorff Distance

When looking for a distance measure to find the similarity between data points, a well established and accepted approach in computer vision and image processing shows up, known as the Hausdorff Distance. This distance is a comparison measure between two point sets. It measures the extent to which each point from one set lies near some point of the other

set. As such, the Hausdorff distance can be used to determine to which degree two points are similar to each other. A more formal definition of the Hausdorff distance are as follows;

Given two finite point sets $A = \{a_1, ..., a_i\}$ and $B = \{b_1, ..., b_j\}$ (e.g. data points from a source and a target dataset), the Hausdorff distance is defined as:

$$H(A, B) = max(h(A, B), h(B, A)) \tag{3}$$

where

$$h(A, B) = \frac{maxmin}{a \epsilon A b \epsilon B} ||a_i - b_j|| \tag{4}$$

The function $||a_i - b_j||$ is the Euclidean norm of the points a_i and b_j (Huttenlocher et al., 1993). The function $h(A, B)$ in equation 4 is also referred to as the 'directed' Hausdorff distance from A to B. First it identifies the point of $a_i \epsilon A$ that is the furthest away from any point of B. It then measures the distance from a_i to its nearest neighbor in B, using the Euclidean norm (Huttenlocher et al., 1993). As given by equation 3 the operation is also performed with $h(B, A)$, and whichever of the two distances are greater will be selected, thus enabling it to find the discrepancy between the two sets.

However, a limitation to this measure is introduced. As the measure takes into account the farthest away points, it is extremely sensitive to just a few, or even a single outlying point of either A or B. This can skew the distances between points immensely resulting in the points being allegedly dissimilar, even though the points might be fairly similar.

A solution to this problem was given by Dubuisson and Jain (1994) who did an investigation of 24 different versions of the Hausdorff distance. They found several candidates that performed equally well or better than the traditional Hausdorff and finally concluded upon a single well functioning version, which they named the *'Modified Hausdorff Distance'* or HMD for short (Dubuisson & Jain, 1994). The HMD measure is defined as:

$$h_{mod} = \frac{1}{|A|} \sum_{a_i \epsilon A} \frac{min}{b \epsilon B} ||a_i - b_j|| \tag{5}$$

Here $|A|$ is the number of points in A. It decreases the impact of outliers by taking the average of the single point distances, making it more robust than the traditional Hausdorff distance.

Based on the state of the art methods presented in the article by Howard & Ruder (2018), further investigation on these method will be conducted throughout this paper. Experiments regarding gradual learning rate for each layer will be carried out to see how well this approach increases performance in an image classification setting and whether it is possible to find a good estimate of said hyperparameter. Additionally, the Modified Hausdorff distance measure will be used to describe the similarity between datasets, and help quantify the distance between two datasets. In the following chapters additional analysis will be conducted to help understand the problems that may occur when working with machine learning.

4 Dimensionality Reduction

Machine learning problems often contains thousands or even millions of features for each data point. Achieving a good fit with a model on such high-dimensional data are difficult and can take an extremely long time. This implication is often referred to as the curse of dimensionality. Géron (2017) presents a theoretical example that explains this implication very well. Consider a random point in a unit square (1x1 square), it will have about 0.4% chance of being located less than 0.001 from the border. In other words, the probability that this random point is "extreme" along any of the two axis, are very small. Whereas for a random point in a 10,000-dimensional unit hypercube (1 x 1 x ,. . . , x 1 cube), this probability is greater than 99.999999% as most points in the high-dimensional hypercube are close to any border. Furthermore, Géron (2017) also explains how dimensions affect the distance between points in space. Given the example of a unit square, the distance between two random points will on average be around 0.52. If the dimensions are incremented and we take two random points in a unit cube (1x1x1 cube) the average distance is increased to around 0.66. This implies that there is a risk of high-dimensional datasets being sparse, making training instance likely to be far from each other. Subsequently, new instances are thus more likely to be far away from any training instances, making predictions less reliable as it is more prone to overfitting.

Géron (2017) suggest that a theoretically simple solution to the curse of dimensionality could be to increase the amount of instances in the training dataset to reach a sufficient training density. Yet, in practice this approach is not viable, as the number of training instances required increases exponentially with the number of dimensions, to reach a given density. With just 100 dimensions (much less than the 784 dimensions in the MNIST dataset), the amount of training instances needed is more than all atoms in the observable universe, for training instances to be within 0.1 of each other on average - assuming they were spread out uniformly across all dimensions.

Fortunately, real data is often confined into a region of the space having much lower effective dimensionality. Thus, the directions over which important variations in the data points occur are therefore confined within this space. In addition, real data will typically also exhibit some smoothness properties (at least locally) so that in most cases, small adjustments on the input data point, will only affect the true output slightly (Bishop, 2006,

37). When again examining the real dataset, MNIST, these two properties can been seen. Pixels near edges of the 28 by 28 images represent very little to none of the dataset's variance. Further, slightly changing some pixel values in an image will most likely not ruin its semantic meaning.

4.1 Principal Component Analysis

Principal Component Analysis (PCA) is considered to be the most popular dimensionality reduction algorithm, as it is deterministic and allows for data exploration and visualization by transforming the data onto a lower dimensionality space (Géron, 2017). The general idea behind PCA is to identify a hyperplane that is located closest to all data-points in a dataset, and then project the data onto that plane. Before being able to project the data onto the lower dimensional hyperplane, one must first select the right plane to do so. This choice is guided by the variance of the dataset, as the axis that preserves the largest amount of variance will most likely result in the least information loss when projected, and is therefore the optimal candidate (Géron, 2017). By applying PCA on a dataset it identifies the axis that account for the largest amount of variance. Then it finds a new axis, orthogonal to the first one, that describes the second largest amount of the variance, i.e. the remaining variance. It will repeat this process for as many axes as the number of dimensions that are to be found in the dataset (e.g. 784 in the MNIST dataset). The unit vector that defines these axes are known as principal components, hence the name of the method (refer to figure 5 for an example of principal components).

Once the principal components are identified, one can then reduce the dimensionality down to n number of dimensions, by projecting it onto the hyperplane defined by the n number of principal components. Thus it preserves as much variance as given by those dimensions (Géron, 2017). The number of dimensions one decides to reduce down to is arbitrary, and is often chosen in regard to the task at hand. If one desires to visualize or explore the data, then two or three dimensions are useful as they can be represented on a graph. However, if it is not for visualization, then a common rule of thumb is to use enough dimensions to account for roughly 95% of the variance (Géron, 2017).

A useful outcome of PCA is that it merely scales the space in which the data points are located. This means that the in-between distances of the data points still remains true to the origin. Additionally, as mentioned

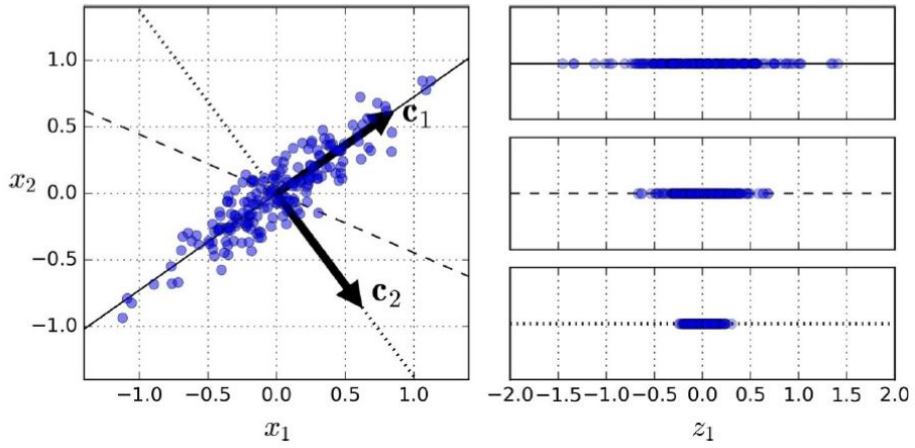


Figure 5: *Left: The principal components (c1, c2) of a dataset. Right: The amount of variance the principal components account for if chosen as the projection axis. The solid line, c1, accounting for most of the variance, while the dotted line, c2, accounts for the least variance (Géron, 2017, pp. 210-220).*

by Géron (2017) the distance between points changes coherently with the number of dimensions chosen and should therefore be kept consistent to have true measures.

4.2 t-distributed stochastic neighbor embedding (t-SNE)

Another method that is widely used to explore and visualize high dimensional data is the *t-SNE* method (Maaten & Hinton, 2008). This method allows one to reduce a high dimensional dataset down to two-, or three-dimensions to create compelling plots. The t-SNE algorithm models the probability distribution of the closest neighboring points for each point in the dataset. In the original high dimensional space this is modeled as a gaussian distribution, whereas in the lower two- or three-dimensional output space it is modeled as a t-distribution (Maaten & Hinton, 2008). The end goal of this procedure is to find a mapping onto the lower dimensional space that minimizes the difference between these two distributions over all data points in the dataset. Using the t-distribution compared to a gaussian in the output space helps spread the data points more evenly. An additional feature of t-SNE which is a tunable parameter that controls the fitting of the data is called *perplexity* (Maaten & Hinton, 2008). Roughly speaking, the perplexity is equivalent to the number of nearest neighbours that the algorithm considers when reducing the data points from the original space to the output space. Maaten & Hinton (2008) argues that the performance

of t-SNE is rather robust to change in perplexity, but having values ranging between 5 and 50 seems to give the best results.

Even though t-SNE is popular due to its ability to find and visualize structures in high dimensional data, one may be prone to misinterpreted these visualizations. This is an issue that has been presented by Wattenberg et al. (2016) on their online and interactive page *'How to Use t-SNE Effectively'*. The following paragraph will discuss the possible misinterpretations of t-SNE and use some of the well illustrated examples provided by Wattenberg et al. (2016). One of the first noticeable things about t-SNE is the perplexity parameter. As stated by Maaten & Hinton (2008) then the values between 5-50 should provide usable and reliable results. As can be seen on figure 6, then data points do indeed cluster well in this perplexity range, whereas a perplexity of 2 or 100 shows very different behaviour. A very low perplexity will often cause the local variations to dominate while a high perplexity will have the opposite effect being more global. Wattenberg et al. (2016) mentions that reason for the two clusters to merge, as seen on the most right handed image in figure 6, are that the perplexity value should be smaller than the number of points - a pitfall that should be avoided when implementing t-SNE.

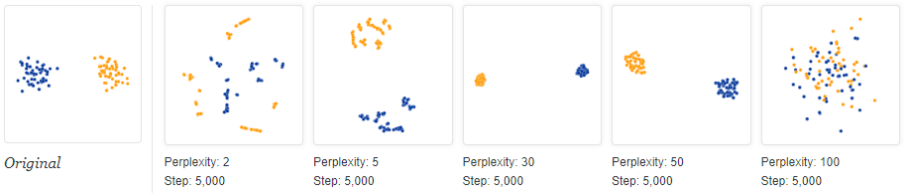


Figure 6: *Displays the original data containing two color coded clusters as well as their behaviour using 5 different perplexity values (Wattenberg et al., 2016, retrieved from https://distill.pub/2016/misread-tsne/.)*

It is obvious that the perplexity value has a tremendous impact on the way the data clusters and setting the value can be a bit devious.

Another part of t-SNE that can be misleading is the cluster sizes. Naturally if a cluster size is dense the data points will appear closer to one another and the standard deviation of that cluster will be low compared to a more sparse cluster. In figure 7, the blue cluster is 10 times more dispersed than the yellow in the original data.
When subjected to t-SNE the two clusters seems about the same size. This indicates that the relative size of clusters are lost during the procedure, as

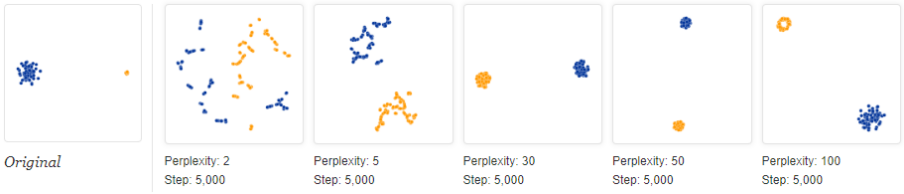


Figure 7: *Relative cluster size for different perplexities (Wattenberg et al., 2016, retrieved from https://distill.pub/2016/misread-tsne/).*

the nature of the t-SNE algorithms expands dense clusters, while contracting sparse ones, ultimately equalizing the size (Wattenberg et al., 2016). Thus the size of clusters in a t-SNE plot has no meaning and should not influence any decision making.

Similar to their size, the distance between clusters may not have any substance after running t-SNE. In figure 8 its seen that the original data have three clusters, with the green cluster being much more distant than the blue and yellow.

Figure 8: *Cluster distances for different perplexity values (Wattenberg et al., 2016, retrieved from https://distill.pub/2016/misread-tsne/).*

At perplexity 50 one can see that the global structure is similar to that of the original data, whereas the other perplexity values show somewhat equal distances between all three clusters. However, if more data points are added, then using a perplexity of 50 does not result in the same structure, meaning that one has to fine-tune the perplexity value in regard to the number of points to recreate the original structure.

There are additional valid and interesting subjects that practitioners should be aware of to not misinterpret the plots provided by t-SNE, however they are out of scope for this project. Please refer to Wattenberg et al.'s (2016) webpage https://distill.pub/2016/misread-tsne/ for further reading and testing with a simple and intuitive interface. It is apparent from the discussion on t-SNE that some substantial pitfalls exists with this method

and using it for other than visualization may not yield the results one would assume.

5 Hyperparameters

When working with machine learning algorithms and the construction of neural networks, it is essential to be able to control the behaviour of the learning algorithm for it to achieve a good performance. Such a controllable setting is known as a *hyperparameter*. For context let us first mention the difference between *parameters* and *hyperparameters*. When training a network towards a given task using a training dataset, we iteratively update the parameters (weights and biases) of our learning algorithm, through the use of back propagation, until convergence is achieved. Thus the values of the parameters are adapted by the learning algorithm itself. Hyperparameters on the other hand are not adapted by the learning algorithm, but are rather defined beforehand and used to ultimately determine how the parameters evolve or behave during training (Goodfellow et al., 2016). Examples of hyperparameters could be the learning rate α , number of hidden layers L, number of neurons in a layer n, batch size, drop-out amount, or even the activation function (e.g. ReLU). The reason for these settings to be chosen as hyperparameters, that a learning algorithm does not need to learn, is that they are difficult to optimize. Additionally, the hyperparameter oftentimes refers to settings that control the model capacity or architecture, e.g. size, width and activations (Goodfellow et al., 2016). If these settings are not being used as hyperparameters but rather as parameters that learns on the training dataset, it would most likely choose the maximum possible model capacity for the specific task, every time, ultimately resulting in overfitting.

In their article *"Collaborative hyperparameter tuning"* Bardenet et al. (2013) describe that previous it has been a devious task to set good hyperparameters. Additionally one needs to manually change the hyperparameter if and when the network does not seem to operate well. To find good hyperparameters however, is more easily achieved by practitioners who have prior knowledge and a better heuristic understanding of the task at hand. Bardenet et al. (2013) further mentions that heuristic experts are better at finding connections between similar tasks and can thus choose a more generalized value for the hyperparameter that may benefit across multiple tasks. However, more recent advanced to hyperparameter tuning makes it possible to apply a nested learning method in which one learning algorithm can be used to find the best possible hyperparameters for another learning algorithm (Goodfellow et al., 2016). There exists various method to achieve optimized hyperparameters and the following subsections will elaborate on a pair of the most popular methods.

5.1 Frequency based optimization

As mentioned above, it has traditionally been costum to manually change the hyperparameters. To supplement this way of finding hyperparameters, a method known as Grid Search have earlier been used in parallel to help automate the process (Bergstra and Bengio, 2012). The algorithm is very straightforward and easily implemented, as one simply have to predefine a set of different hyperparameter values, train a model for each possible combinations hereof, and finally selecting the hyperparameter values that leads to the best performing model. This is a considerable exhaustive method to find the right hyperparameters, brute forcing all possible combinations of values, which seemingly will take quite a substantial amount of time. For context, let us imagine that we want to optimize just 4 hyperparameters, using 10 different values for each parameter. Thus the number of evaluations one have to carry out would be 10 to the power of 4 (10^4), which equals to 10.000 evaluations. Based on the time it takes to train a network, which often is time consuming in its own, one would essentially have to wait weeks, months or even years before it would have found the optimal hyperparameters. And the optimal hyperparameters using grid search would still be limited to those predefined values, meaning that the truly optimal value may not be represented in the search space, a consequential shortcoming to note.

5.1.1 Random search

Another method that can be applied to find optimal hyperparameters, which is an extension to the grid search, is the Random Search method. Rather than hard coding or predifining the values one wants to investigate and then exhaustively try all possible combinations, with random search one can specify a range of values for each hyperparameter. Within these various ranges a random value will be selected and the the combinations thereof will then be evaluated. This process is then iterated for as many trials as specified by the practitioner. In their article 'Random Search for Hyperparameter Optimization' Bergstra and Bengio (2012) experiment with random search and finds it to be equally comparable to grid search on performance. However, the amount of trials needed to find hyperparameters that results in good performance are significantly lower than those of a normal grid search making it the more prominent method to use. The authors argue that if the close-to-optimal region are indeed present in the search space, then a random search with around 40-60 trials and/or iterations, have a significant high probability ($\sim$ 95% confidence) of finding

that region.

Grid search as well as random search are both objective frequency based optimizers, that needs a predefined search space in order to find the most optimal hyperparameters. This pose a limitation for practitioners that do not have much experience or knowledge in the field as the space may not be intuitive. The methods only takes into consideration the best possibilities that it came across and does not allow for much exploration of good or near-optimal regions of the search space. The next method that will be presented is more subjective in the matter of exploring the search space where a potential good region is found. This method is known as Bayesian optimization and will be elaborated upon in the following section.

5.2 Probabilistic based optimization

In bayesian optimization the goal is to find the minimum loss of a function $f(x)$ on a given dataset X, as is the goal with any other optimization algorithm. However, what makes bayesian optimization different from the previously mentioned optimizers, is that it constructs a probabilistic model of the function $f(x)$ and then explore this model in order to give a qualified assumption about where in X it should evaluate the function next (Snoek et al., 2012). Thus, the idea behind the bayesian optimization is to use all of the prior observations available of the function $f(x)$, to help determine the next and optimal point to sample in X, without having to rely solely on the local gradient. The trade-off is between exploring the search space (in this case X), making sure to investigate the most relevant points, and exploiting the settings for this space to find the most promising parameters within it. The result thereof is much fewer iterations but at the cost of more computational power needed, compared to Random and Grid search (Snoek et al., 2012). Thus, a procedure of finding the minimum loss of a difficult non-convex function $f(x)$ is feasible.

To implement bayesian optimization, there are two overall choices that a practitioner's must take. The first one is to select a prior over functions that is able to make assumptions about the function being optimized. The common choice is the Gaussian Process Prior, as its structure and shape of the underlying function is rather flexible and tractable (Snoek et al., 2012). A Gaussian Process (GP) is the generalization of a gaussian distribution to a distribution over functions instead of random variables. Similar to a gaussian distribution being specified by its mean and variance, a GP is

specified by its mean function and covariance function. As an example; for a set of data points $x_n = \{x_1, ..., x_n\}$ we assume that the values of the loss function $f_n = \{f(x_1), ..., f(x_n)\}$ can be described by a multivariate gaussian distribution, thus we have:

$$f_n \sim \mathcal{N}(m(x_n), K) \tag{6}$$

where $m(x_n) = \{m(x_1), ..., m(x_n)\}$ is the mean function and $K = k(x, x')$ is the covariance function. Therefore, instead of returning a scalar $f(x)$, a GP returns the mean and variance of a normal distribution over the possible values of the loss function f at x.

The second choice one has to make is to select an acquisition function to find the best point to evaluate next. This is a function of the posterior distribution over the loss function f, that describes the utility for all values of the hyperparameters. The values that are determined to have the highest utility will be selected as the point for which the loss function will be computed next (Snoek et al, 2012). There exists different kinds of acquisition functions and Snoek et al (2012) mentions that the function known as *Expected Improvement*, also denoted as EI, have shown to be popular and reliable. It can be defined as follows:

$$EI(x) = \mathbb{E}[max(0, f(x) - f(\hat{x}))] \tag{7}$$

Where $\hat{x}$ is the current optimal set of hyperparameters. The reason for using the maximum argument is that it will provide the point that it expects to improve the loss function f, the most. Furthermore, the expectation of EI can be computed under the Gaussian Process Prior, to give insights to what sort of values that will result in higher improvements (Snoek et al., 2012). This is achieved by its predictive mean function μ and predictive variance function σ and can be written as:

$$EI(x) = \begin{cases} (\mu(x) - f(\hat{x})\frac{\mu(x)-f(\hat{x}}{\sigma(x)}) + \sigma(x)\frac{\mu(x)-f(\hat{x}}{\sigma(x)}, & \text{if } \sigma(x) > 1 \\ 0, & \text{if } \sigma(x) = 0 \end{cases} \tag{8}$$

What can be derived from equation 8 is that either the EI will be high when the posterior value of the loss $\mu(x)$ is higher than the current best value of $f(^x)$, or EI will be high when the uncertainty $\sigma(x)$ around the point x is high. This is seemingly intuitive as if one maximizes the EI, then it will either sample from points where it expects a higher value of the function f,

or else it will sample points in a region of the function f that has not been explored yet.

The use of Bayesian optimization to find well functioning hyperparameters for a model is seemingly an efficient and favourable method that practitioners can apply. However, an ironic dilemma occurs, as one needs to decide upon hyperparameters for the Bayesian optimization both in regard to which covariance function to use as well as the settings of the gaussian process. This results in an endless loop, that magnifies how empirical frameworks, tests and results can be of great use, to find both a common ground or baseline for different methods and approaches, as well as provide an indication of what values to initialize the hyperparameters with.

6 Optimizers

In the previous section, parameters of a network was briefly introduced as the weights and biases, which are updated during training through the use of backpropagation. Backpropagation by gradient descent is one of the most common algorithms used to optimize neural networks, and many variants of gradient descent has been implemented to further improve the method (Ruder, 2016). Throughout this chapter an overview of current and popular gradient descent optimizers will be introduced and elaborated upon.

6.1 Traditional Gradient descent

The idea behind gradient descent in machine learning is to minimize a loss function in regards to the models parameters. Assume that the coefficients of the parameters have some value θ, then the objective function can be denoted as $f(\theta)$. The derivative of the objective function should then be calculated to find the slope of the function. This allows one to update the parameters in the opposite direction of the gradient, following the slope down-hill towards its minima. Additionally, a learning rate should be specified to determine how big a step is taken in the downhill direction in order to reach said minima. Thus the algorithm for gradient descent are as follows:

$$\theta = \theta - (\eta \cdot \nabla_\theta f(\theta)) \tag{9}$$

6.1.1 Batch gradient descent

There are three variations of gradient descent that differ from one another based on how much data is used to compute the gradient. Depending on the amount of data used, a trade off between how much time it takes to perform an update and how accurate the parameter update will be, is introduced (Ruder, 2016).

The traditional gradient descent algorithm, also referred to as batch gradient descent, was presented above in equation 9. However, a shortcoming of this algorithm is that one has to compute the gradients for the entire training dataset to perform a single update. Thus, it can be an extremely slow operation if the dataset is considerable large, as well as being close to unmanageable if the dataset is is not able to fit in memory (Ruder, 2016).

6.1.2 Stochastic gradient descent (SGD)

To resolve the intractability of batch gradient descent, the stochastic gradient descent (SGD) can be applied. Rather than making a parameter update at the end of each batch, the update is performed for each training instance with regard to their respective label. As SGD performs much more frequent updates with higher variance it causes the objective function $f(\theta)$ to fluctuate more. This fluctuation introduces both pros and cons to the method. The pros of SGD is that learning can be performed much faster compared to batch gradient descent, and it enables the possibility to jump to new and potentially better local minimas. The downside to this functionality on the other hand, is that SGD may impede convergence to the exact minimum as it can overshoot the goal repeatedly (Ruder, 2016).

6.1.3 Mini-batch gradient descent

The last variation of traditional gradient descent, combines the most useful parts from batch and stochastic gradient descent, and is known as mini-batch gradient descent. This works by performing an update for every subset of training instances. Thus the variance of the parameters updates are reduced, resulting in a less fluctuating convergence, and furthermore, allows the practitioner to choose the size of the mini-batch. This makes mini-batch gradient descent a common choice for training neural networks in practice (Ruder, 2016).

However, mini-batch gradient descent still has some challenges, which relates to choosing the right settings, or hyperparameters, which earlier discussed, can be a a devious task. One recurring difficulty is choosing the right learning rate, as a too low learning rate will increase the time it takes to convergence, while a too high learning rate may hinder convergence or even result in divergence (Ruder, 2016). Additionally the learning rate schedule, e.g. to use annealing (as presented in section 3.2.1, p. 12), to help overcome both flat saddle points and bad local minimas, has to be defined beforehand. To overcome these shortcomings, various optimization algorithms have been developed to adapt the learning rate in certain situations. The articles written by Singh et al (2015) and Ruder (2016) encloses a great overview of the different optimizers, and the upcoming section will follow and impart their work.

6.2 Gradient Descent Optimizers

One of the earlier methods to adapt the learning rate is known as *momentum* (Qian, 1999). This method works by increasing the learning rate for parameters which gradients constantly points in the same direction, while decreasing the learning rate for parameters which gradients rapidly change (Singh et al, 2015). Thus, faster convergence can be achieved. However, another problem may occur when using momentum. As momentum accumulates when going down hill it will often be rather high close to the goal point, i.e. the minima, and do not know that it should slow down, potentially causing it to miss the minima entirely.

To counter this problem the *Nesterov accelerated gradient*, aka *NAG* (Nesterov, 1983), was invented to give momentum this functionality of knowing when to slow down before going uphill again (Ruder, 2016). By predicting the gradient for the next step it will change the learning rate for the current step, based on that prediction. Therefore, if the gradient is predicted to be higher for the upcoming step, it will increase the learning rate, whereas if the prediction shows a lower gradient the learning rate will decrease (Singh et al, 2015). With NAG it is possible to adapt the updates based on the slope of the gradient, however, it is in regard to all the function parameters.

Further development have sought to adapt the updates to each individual parameter, instead of collectively, to make bigger or smaller updates, depending on their importance (Ruder, 2016). This has been achieved with the algorithm known as *Adagrad* (Duchi et al., 2011), which adapts the learning rate to individual parameters based on its frequency. Bigger updates are performed to infrequent parameters while lesser updates are performed to frequent parameters. It does so by using a different learning rate for every parameter, at a given time step, based on the past gradients that were computed for that parameter (Singh et al, 2015). Thus it becomes obsolete to manually tune the learning rate. The main disadvantage of Adagrad however, is that the learning rate is always decreasing. This is due to the the accumulation of the gradients over all iterations, a sum in the denominator that continues to grow throughout training. Eventually the learning will become so small that it will be unable to acquire further knowledge, hindering the numbers of iterations that is useful for training (Ruder et al, 2016).

To accommodate for the constant decreasing learning rate an extension to Adagrad, called *Adadelta* (Zeiler, 2012), has been developed that seeks to

resolve just that. Rather than accumulating all the past gradients, the window in which the gradients are summed is restricted to a fixed size (Singh et al., 2015). Furthermore, the sum of gradients in this fixed size, are defined as a decaying average of all the past squared gradients, meaning that the running average at a given time step, depends on the previous average as well as the current gradient (Ruder, 2016). This counteracts the denominator from becoming extremely small, and allows for further updates of the parameters, without the diminishing return of reaching too many iterations.

The improvements from traditional gradient descent and up until Adadelta have involved calculating the momentum for faster convergence, compute adapting learning rates for individual parameter as well as preventing vanishing learning rates. To further this work a method known as *Adam* (Kingma & Ba, 2014) have been developed that in addition to computing the adaptive learning rates for each parameter also store the momentum changes for each of them separately. By calculating the first moment, the mean, and the second moment, the uncentered variance, of the gradients respectively it is possible to update the parameters individually, similar to the approach in Adadelta. This method empirically shows to outperform Adagrad and Adadelta in practice (Ruder, 2016).

What can be seen from literature and the history of gradient descent is that the improvements over the years have resulted in clever and efficient ways of optimizing the learning rate as presented above. It is apparent that the parameters of a network behave differently and some needs to be updated more than others, due to their importance and/or frequency. This notion is quite similar to how different layers of a network capture different features and should thus be trained to different extents (as presented in section 3.1, p. 7). However, to our knowledge, only one implementation of an optimizer able to update the learning rate layer-wise has been introduced, namely, the discriminative fine-tuning method proposed by Howard & Ruder (2018). In regard to transfer learning the goal is to train a well performing network on a target task, with only little available data, by utilising prior information captured from a pretrained network trained towards a different task. To which extent the network should be trained depends very much on the relatedness of the source and target tasks, which in turn could be objectively quantified by utilizing the Modified Hausdorff Distance. In the following chapter we will present the experimental setup of our test including the procedure of the experiments.

7 Experimental Setup

The main objective for our empirical experiment is twofold; (1) to investigate the effectiveness of the promising state-of-the-art approach presented by Howard & Ruder (2018) with Learning Rate Finder, Warm Restarts, Gradual Learning Rates and Cosine Annealing, in a image classification context, (2) examine the potency of using the Modified Hausdorff distance measure to predict the knowledge transferability of one task to another.

7.1 Specifying datasets and domain adaptation

The two datasets used in the experiments is a subset of the balanced EMNIST dataset (Cohen, 2017) and the Cifar10 dataset (Krizhevsky, 2009). The balanced EMNIST dataset is an extended version of the original MNIST dataset, containing images of both handwritten digits and alphabetic characters. The images are grayscale with a pixel size of 28x28 resulting in 784 pixels per image (i.e number of dimensions). It has a total of 47 classes with 3000 images per class. The Cifar10 dataset consists of 60000 color images with a pixel size of 32x32x3, resulting in 3072 pixels per image. It has a total of 10 classes with 6000 image per class representing airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks.

To generate the source and target datasets for the experiment, we extracted four random samples, two from the EMNIST dataset and two from the Cifar10 dataset. These samples will from here on out be referred to as Emnist01, Emnist02, Cifar01 and Cifar02 respectively. Each sample consist of 4 classes with either 3000 or 1200 images in each class, dependant on whether it is used as the source or target task. The reasoning behind using a reduce number of training points for the target tasks, is to simulate a real world transfer learning scenario of having less training data available. In table 7.1 an overview of each random sample and its corresponding classes can be seen.

In order to accommodate for the inconsistent image sizes (Emnist being 28x28x1 and Cifar10 32x32x3) two different domain adaptations has been performed, to transform all instances into a shared dimensional space. First, an asymmetric transformation is done in the form of changing the EMNIST dataset by adding 2 extra color channels with identical pixel values, going from a grayscale 1 dimensional to a RGB 3 dimensional color space. Second, a symmetric transformation of both image datasets are performed by rescaling all instance into a 224x224x3 image size.

Sample names	**Emnist01** (Source)	**Emnist02** (Target)	**Cifar01** (Source)	**Cifar02** (Target)
Class labels	E, M m, U u, n	0, F, N, V v	Plane, Cat, Horse, Ship	Car, Bird, Dog, Frog
Training sample size per class	2400 images	600 images	2400 images	600 images
Validation sample size per class	600 images	600 images	600 images	600 images

Table 1: *The EMNIST and Cifar10 samples with their corresponding classes and sizes. Sample *01 and *02 are always used as source and target tasks, respectively. For the source task 80% of the sample (9600 images) is used as training and the remaining 20% of the sample (2400 images) is used for validation. For the target task 50% of the sample (2400 images) is used as training and the remaining 50 % of the sample (2400 images) is used for validation. Having an equal amount of images for validation ensures consistency between the performance of source and target tasks.*

7.2 Distance Measure of Datasets

The modified hausdorff distance is calculated for each of the four dataset comparisons, and saved for later examination when all networks has been trained. The measure is performed on the source and target training sets, generating a scalar value for each class comparison. Before doing the measure, both datasets are scaled to equal size by randomly excluding points from the largest classes, down to the smallest. Further, incremental principal component analysis is performed on the source and target dataset collectively. This is done to decrease every instance in both datasets down to a fixed reduced dimensional space consisting of the 148 principal components. This is implemented with the intent of counteracting the curse of dimensionality, and in practice still keeps about 95.0% of variance from the original data (See appendix E).

7.3 Source and Target Networks

Figure 9 shows an overview of the experimental setup for testing gradual learning rate. The network chosen for this setup is small (considering the standard depth of the currently used conv-networks) with a fairly simple architecture, called Resnet34 (He et al., 2016). The first figure shows the Resnet34 being trained towards the source dataset, in this case the Em-

nist01 sample. This network will serve as our pretrained network. In the second figure, we see the pretrained network being trained towards the new target task, i.e. the Cifar02 sample. The colors are used to indicate how prominent the learning rate is and to visually display how it gradually decreases for each layer during backpropagation. Here purple indicates knowledge gained from the source task, while green indicates knowledge from target. These two figures are merely used as an example of the process (Emnist01 as source, Cifar02 as target), while three additional combinations are tested, namely; Emnist01 (source) → Emnist02 (target), Cifar01 (source) → Emnist02 (target), and Cifar01 (source) → Cifar02 (target).

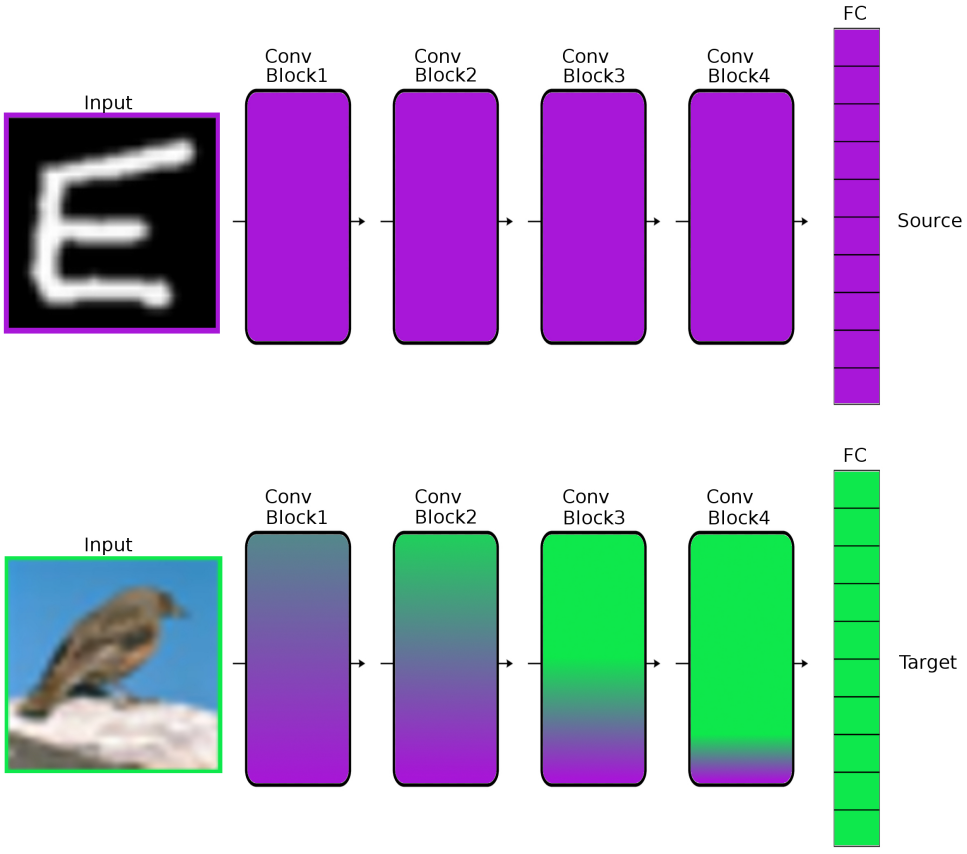


Figure 9: *TOP: An example of the source network trained towards the Emnist01 sample. BOT: An example of the source network being fine-tuned towards a target task (the Cifar02 sample) using gradual learning rate. Here the amount of green represent the learning rate and how it decrease during backpropagation.*

7.4 Choosing the initial learning rate

Choosing the initial learning rate is done utilizing the learning rate finder proposed by Smith (2017). For our experiment it is solely used to estimate a range in which different initial learning rates can be evaluated. It trains the specific model with different learning rates in a broad range starting from a low rate, and outputs the loss value. The model then reverts back to its initial weights before training, and the loss output can be plotted with the purpose of choosing a proper initial learning rate. Figure 10 shows an example of such visualization for the Emnist01 → Cifar02 experiment.

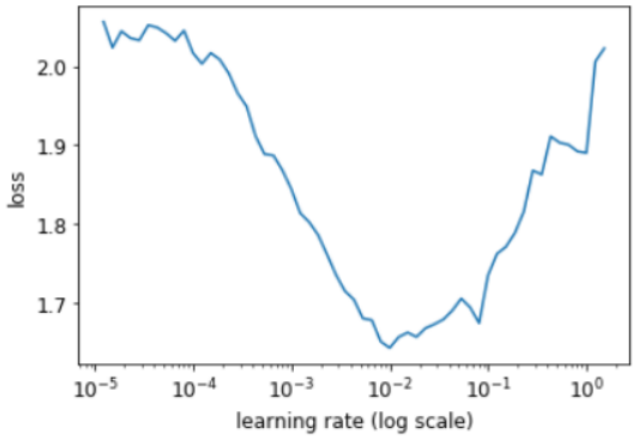


Figure 10: *Showing the learning rate finder output for the Emnist01 → Cifar02 experiment.*

8 Results

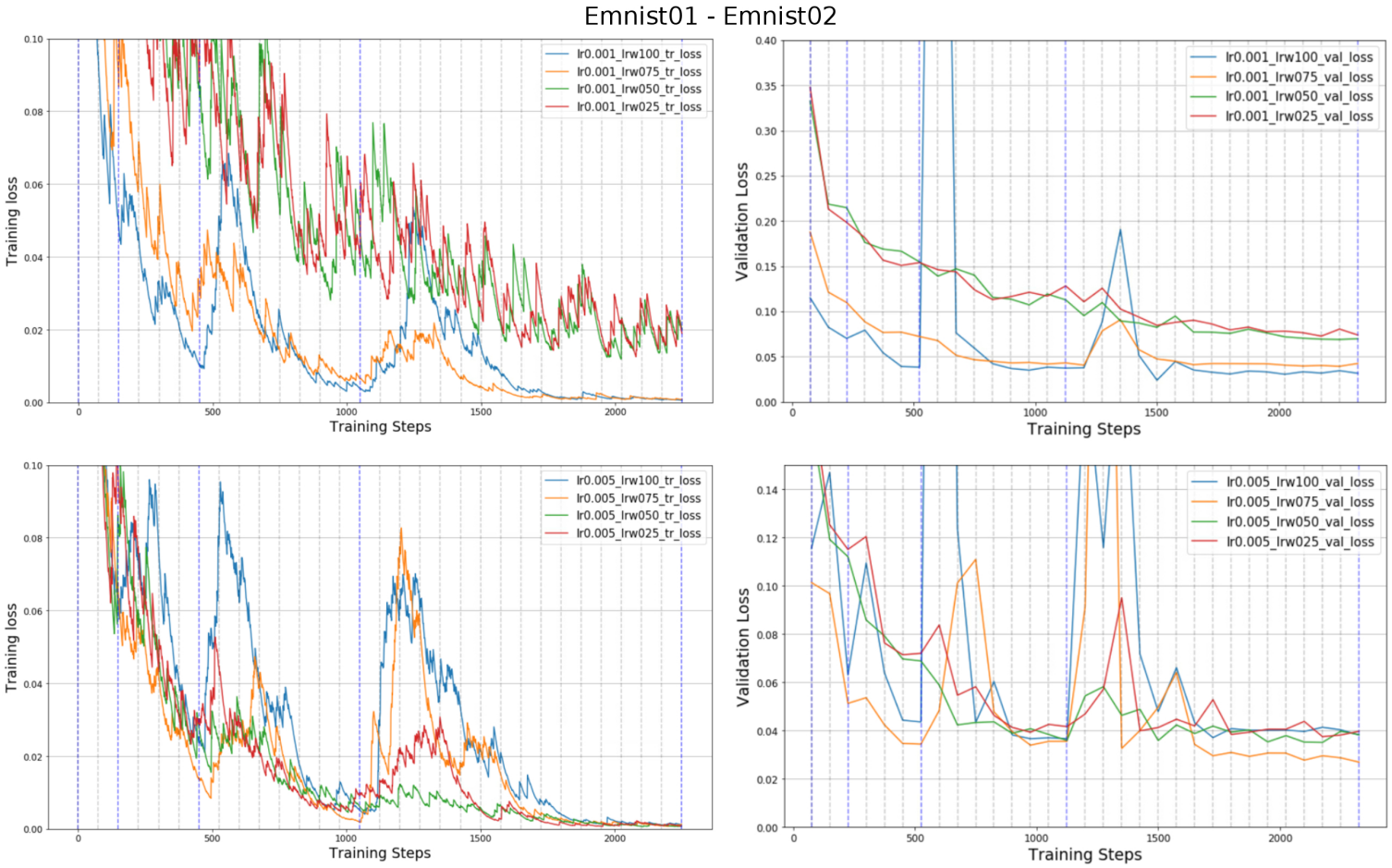


Figure 11: *Results of the network trained on Emnist01 as source and towards Emnist02*

Em01\Em02	**0**	**F**	**N**	**V v**
E	110.44	96.49	118.89	116.78
M m	124.33	117.92	102.82	115.36
U u	100.81	124.53	101.65	87.78
n	104.25	113.73	102.93	114.29

Table 2: *Distance between sample 1 and sample 2 from the Emnist dataset. Highlighted cells shows the shortest distance.*

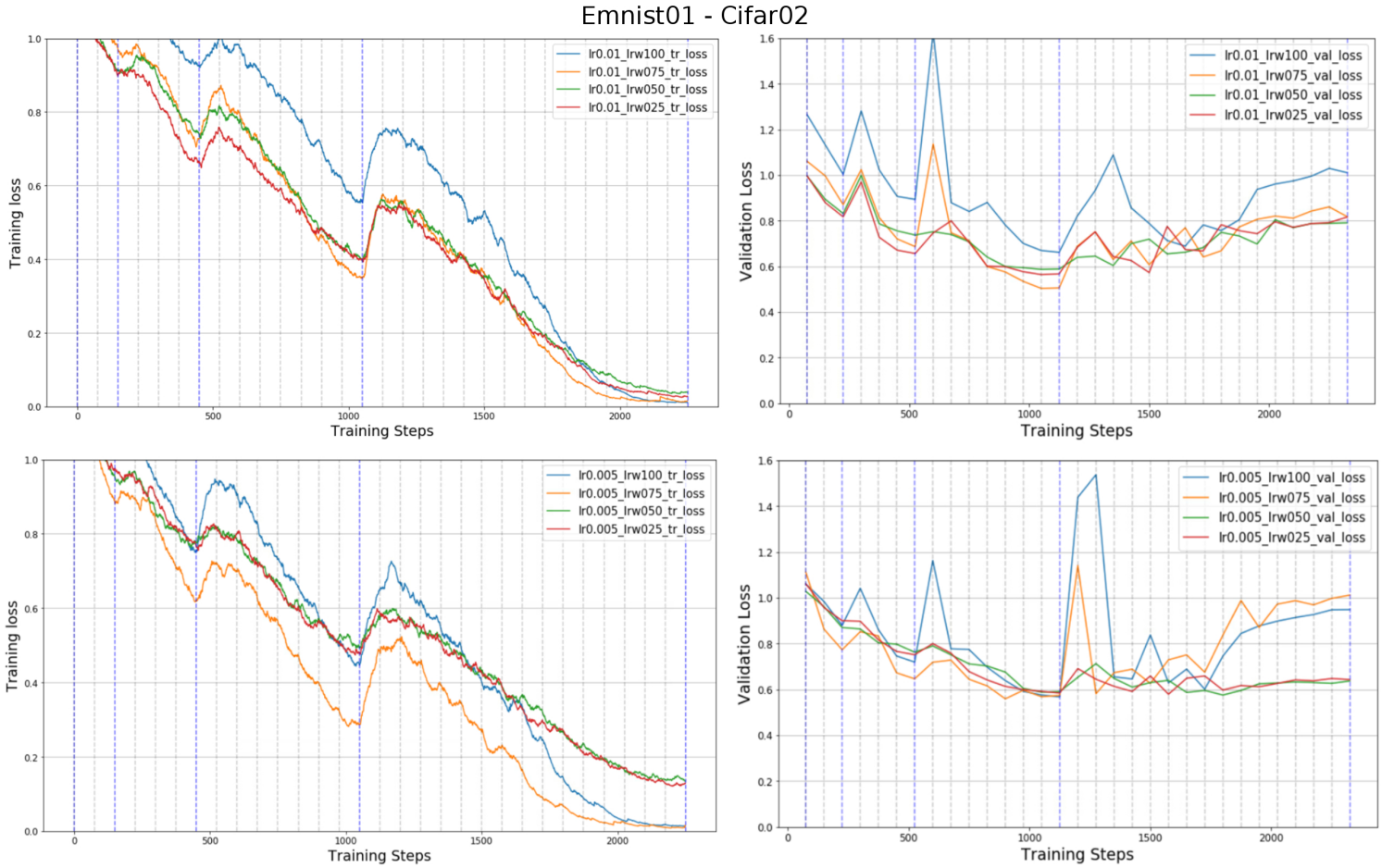


Figure 12: *Results of the network trained on Emnist01 as source and towards Cifar02*

Em01\Ci02	**Car**	**Bird**	**Dog**	**Frog**
E	171.38	167.66	161.50	156.07
M m	170.80	168.19	164.21	156.77
U u	174.81	168.82	163.30	157.96
n	172.35	168.11	162.75	157.13

Table 3: *Distance between sample 1 from the Emnist dataset and sample 2 from the Cifar10 dataset. Highlighted cells shows the shortest distance.*

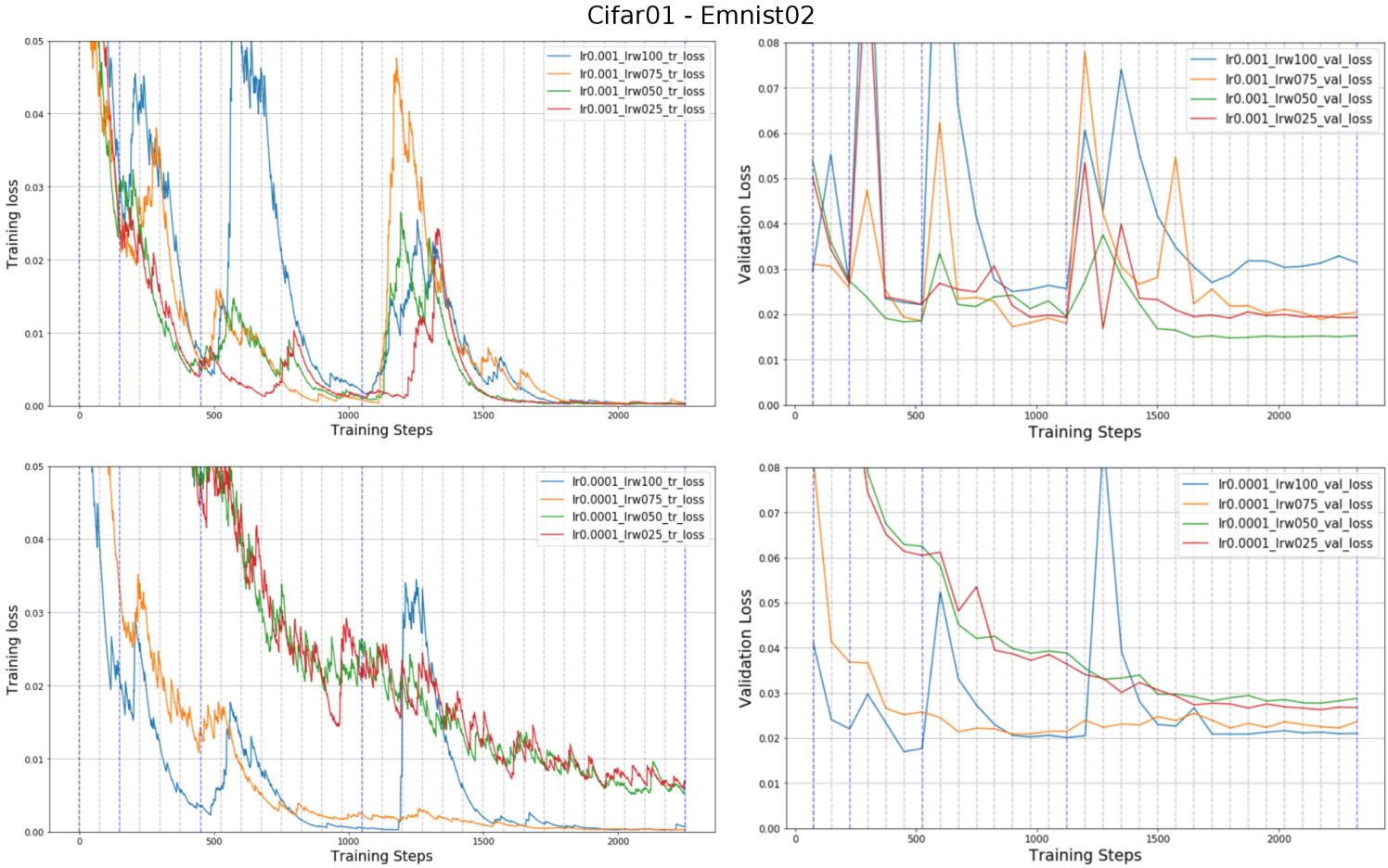


Figure 13: *Results of the network trained on Cifar01 as source and towards Emnist02*

Ci01\Em02	**0**	**F**	**N**	**V v**
Plane	192.33	193.78	194.79	199.53
Cat	167.59	168.93	167.82	170.25
Horse	171.94	172.77	173.14	176.15
Ship	183.07	181.14	184.23	187.63

Table 4: *Distance between sample 1 from the Cifar10 dataset and sample 2 from the Emnist dataset. Highlighted cells shows the shortest distance.*

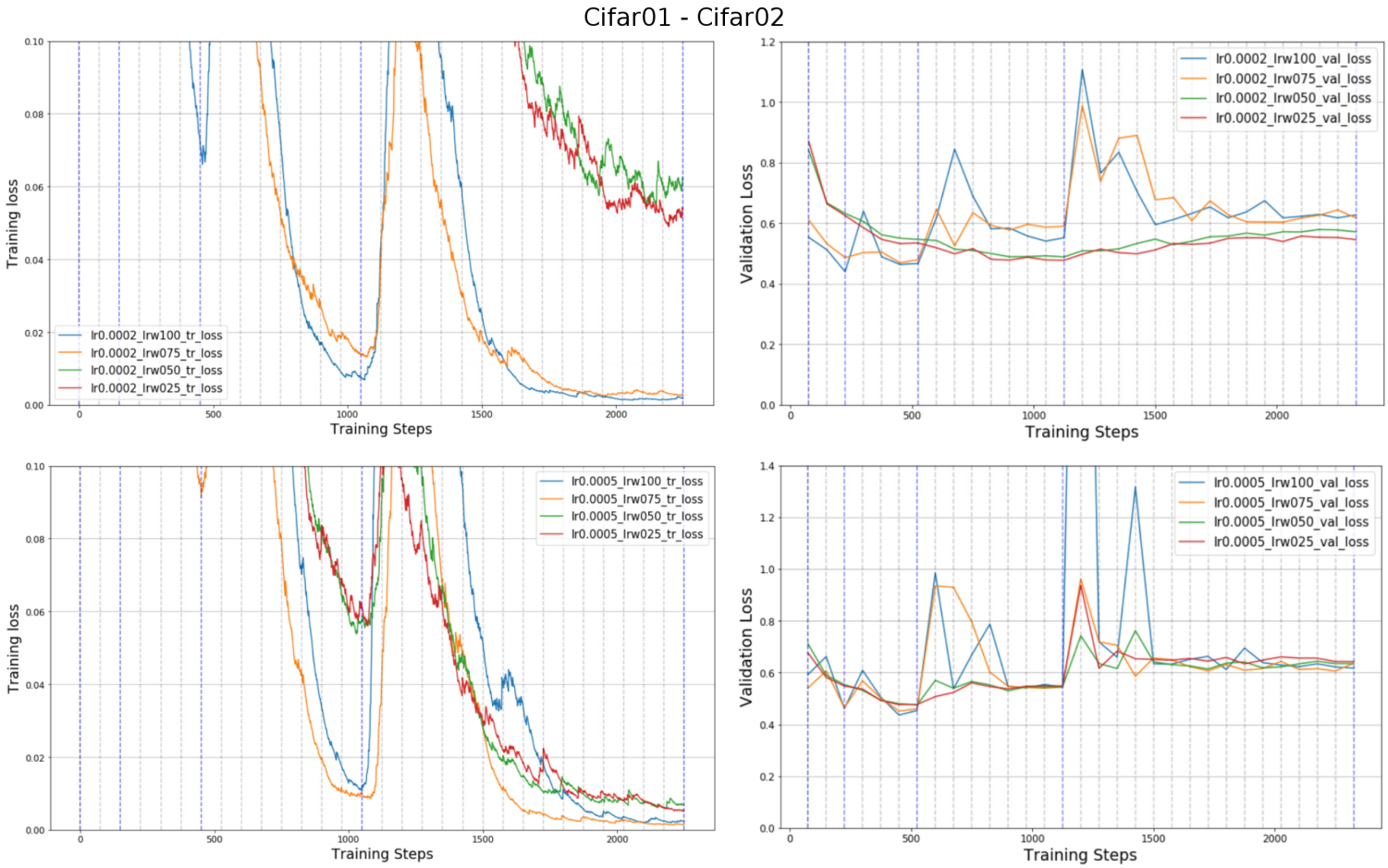


Figure 14: *Results of the network trained on Cifar01 as source and towards Cifar02*

Ci01\Ci02	**Car**	**Bird**	**Dog**	**Frog**
Plane	74.16	62.86	73.30	69.46
Cat	76.63	68.39	68.85	68.18
Horse	76.72	68.53	72.34	69.42
Ship	73.60	68.38	74.30	71.55

Table 5: *Distance between sample 1 and sample 2 from the Cifar10 dataset. Highlighted cells shows the shortest distance.*

9 Discussion

9.1 Evaluating the distance measure

The between distances of samples stemming from the same original dataset versus distances arised from different datasets, produces some intuitively anticipated results. Classes from Cifar10 target dataset (Ci02) are all measured to be more similar to the source Cifar10 dataset (Ci01) than to the source Emnist dataset (Em01), and vice versa. In addition, this occurrence can even be extended to classes coming from the same dataset; see the distance of 'U' (Em01) and 'V' (Em02), which instinctively share similar features versus 'E' and 'V' (Em02). An example of the same phemonenon on cifar10 is 'cat'(Ci01) and 'dog'(Ci02), are measured closer than 'ship' (Ci01) and 'dog'(Ci02). Our proposed method of combining principal component analysis and the modified hausdorff distance mearse, seem to capture the marginal distributions of each dataset quite well. Whether this measure is indeed useful for predicting the knowledge transferability between two task, is discussed in conjugation with the experimental results.

9.2 Insight into CIFAR10 and combined class variance

One notable aspect when comparing the results of the Emnist01→Emnist02 and the Cifar01→Emnist02 experiment, is the end performance difference of the two. The target network trained on cifar01 first, shows a slightly lower validation loss than the one pre-trained on Emnist01. This is similarly the case when comparing Cifar01→Cifar02 and Emnist01→Cifar02. This might indicate that pre-training on Cifar01 produces richer transferable feature weights, even when Emnist01 are considered closer both in a intuitive sense and empirically (by the MHD measure), to the Emnist02 dataset.

The reason for this might be visible when inspecting the scatter plot of both Emnist01→Emnist02 and Cifar01→Emnist02 on the two most dominant principal components (See Appendix E). It shows that the cifar datasets consist of a far greater amount of variance than the Emnist dataset, which might indicate that the overall feature variance is also greater. This may have forced the network to acquire weights that detect a more general sets of low and high level features for the source task. Which, in the end indicates that Cifar10 might share more features with other datasets in general, than the 'invariant' Emnist dataset.

It could be interesting to inspect how much the combined class variance of a dataset influences the effectiveness of the knowledge transferability between tasks. This might turn out to be a better predictor than the marginal distribution difference measured in this study. The motivation for this is that the combined class variance of a given dataset, might reveal a substantial insight into the location on which the true features and label domain (condition 1 and 3) are located in the high dimensional space.

9.3 Confidence of the results

When comparing the different initial learning rates for each experiment, it is seem that it significantly influences how well each learning rate weight performs. Additionally, it is rarely the case that no decrease in the learning rate per layer (100%) or major decrease (25%), produces the highest performing target model. This discovery contradicts Howard & Ruder's (2018) rule of thumb to some extend, as they advice always to use a learning rate decrease 30% per layer when fine-tuning.

One must take into consideration the fairly sparse amount of times each experiment is performed in this study, which is mainly due to the time constraints of this project. However, the confidence interval of all re-training sessions might not be as large as one might initially think, since the target networks are not trained from random weights and biases. This leaves only two random factors when fitting which is the data batch selection and dropouts. It is unknown how much the randomness of those two factors can influence the final performance outcome of the fine-tuned model.

10 Conclusion

The main objective of this report was to investigate the effectiveness of the state-of-the-art transfer learning approaches presented by Howard & Ruder (2018), in a image classification context. This was achieved through empirical experiments conducted on the CIFAR10 and balanced EMNIST datasets. The amount of relevant knowledge transferred from one dataset to the other, was expressed through how well the model performance increased on the target task. One method in particular called the discriminative fine-tuning, that utilizes different learning rates for each layer, shows promising results when applied on all four of the different experiments conducted in this project.

In addition to these transfer learning methods, the potency of a dissimilarity measure, called the Modified Hausdorff Distance, was evaluated as a suitable estimator for how well knowledge of one task transfers to another. This transferability measure was implemented with the motivation of finding a fast, reliable and convenient preprocessing method for deciding the learning rate weight, when fine-tuning a pre-trained network towards a new task. The measure showed no significant correlation between the performance increase of the target task and the distance of the two specified datasets. However, it should be considered as merely measuring one aspect out of the four presented conditions within transfer learning (see section 2, p 4), namely the marginal distribution difference of two given datasets. Further analysis of the experimental results, indicated that the combined class variance might be a promising feature to investigate as a possible measure for knowledge transferability between two tasks.

Acknowledgements

We would like to show our gratitude to the Machine Learning department of Top Danmark A/S, who greatly assisted by providing guidance and covering of expanses on AWS cloud computing services.

Bibliography

[1] Nesterov, Y. (1983, February). A method of solving a convex programming problem with convergence rate O (1/k2). In Soviet Mathematics Doklady (Vol. 27, No. 2, pp. 372-376).

[2] Huttenlocher, D. P., Klanderman, G. A., & Rucklidge, W. J. (1993). Comparing images using the Hausdorff distance. IEEE Transactions on pattern analysis and machine intelligence, 15(9), 850-863.

[3] Dubuisson, M. P., & Jain, A. K. (1994, October). A modified Hausdorff distance for object matching. In Pattern Recognition, 1994. Vol. 1-Conference A: Computer Vision& Image Processing., Proceedings of the 12th IAPR International Conference on (Vol. 1, pp. 566-568). IEEE.

[4] Qian, N. (1999). On the momentum term in gradient descent learning algorithms. Neural networks, 12(1), 145-151.

[5] Bishop, Christopher M. (2006). Pattern recognition and machine learning. New York :Springer

[6] Maaten, L. V. D., & Hinton, G. (2008). Visualizing data using t-SNE. Journal of machine learning research, 9(Nov), 2579-2605.

[7] Krizhevsky, A. (2009). Learning Multiple Layers of Features from Tiny Images. Retrieved from https://www.cs.toronto.edu/ kriz/cifar.html

[8] Pan, S. J., & Yang, Q. (2010). A survey on transfer learning. IEEE Transactions on knowledge and data engineering, 22(10), 1345-1359.

[9] Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. Journal of Machine Learning Research, 12(Jul), 2121-2159.

[10] Bergstra, J., & Bengio, Y. (2012). Random search for hyper-parameter optimization. Journal of Machine Learning Research, 13(Feb), 281-305.

[11] Snoek, J., Larochelle, H., & Adams, R. P. (2012). Practical bayesian optimization of machine learning algorithms. In Advances in neural information processing systems (pp. 2951-2959).

[12] Zeiler, M. D. (2012). ADADELTA: an adaptive learning rate method. arXiv preprint arXiv:1212.5701.

[13] Bardenet, R., Brendel, M., Kégl, B., & Sebag, M. (2013, February). Collaborative hyperparameter tuning. In International Conference on Machine Learning (pp. 199-207).

[14] Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 580-587).

[15] Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

[16] Yosinski, J., Clune, J., Bengio, Y., & Lipson, H. (2014). How transferable are features in deep neural networks?. In Advances in neural information processing systems (pp. 3320-3328).

[17] Azizpour, H., Razavian, A. S., Sullivan, J., Maki, A., & Carlsson, S. (2015). From generic to specific deep representations for visual recognition. In CVPRW DeepVision Workshop, June 11, 2015, Boston, MA, USA. IEEE conference proceedings.

[18] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. (2015). ImageNet Large Scale Visual Recognition Challenge. International Journal of Computer Vision (IJCV).

[19] Singh, B., De, S., Zhang, Y., Goldstein, T., & Taylor, G. (2015, December). Layer-specific adaptive learning rates for deep networks. In Machine Learning and Applications (ICMLA), 2015 IEEE 14th International Conference on (pp. 364-368). IEEE.

[20] Goodfellow, I., Bengio, Y., Courville, A., & Bengio, Y. (2016). Deep learning (Vol. 1). Cambridge: MIT press.

[21] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).

[22] Loshchilov, I., & Hutter, F. (2016). Sgdr: Stochastic gradient descent with warm restarts.

[23] Ng, A. (2016). Nuts and bolts of building AI applications using Deep Learning [Powerpoint slides] - NIPS 2016 tutorial. Retrieved from https://media.nips.cc/Conferences/2016/Slides/6203-Slides.pdf

[24] Ruder, S. (2016). An overview of gradient descent optimization algorithms. arXiv preprint arXiv:1609.04747.

[25] Variables: Creation, Initialization, Saving, and Loading. (2016, April 26). Retrieved from https://www.tensorflow.org/versions/r1.0/programmers_guide/variables

[26] Wattenberg, M., Viégas, F., & Johnson, I. (2016). How to use t-sne effectively. Distill, 1(10), e2, http://doi.org/10.23915/distill.00002.

[27] Weiss, K., Khoshgoftaar, T. M., & Wang, D. (2016). A survey of transfer learning. Journal of Big Data, 3(1), 9.

[28] Cohen, G., Afshar, S., Tapson, J., & van Schaik, A. (2017). EMNIST: an extension of MNIST to handwritten letters. Retrieved from http://arxiv.org/abs/1702.05373

[29] Felbo, B., Mislove, A., Søgaard, A., Rahwan, I., & Lehmann, S. (2017). Using millions of emoji occurrences to learn any-domain representations for detecting sentiment, emotion and sarcasm. arXiv preprint arXiv:1708.00524.

[30] Géron, A. (2017). Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems. " O'Reilly Media, Inc.".

[31] Smith, L. N. (2017, March). Cyclical learning rates for training neural networks. In Applications of Computer Vision (WACV), 2017 IEEE Winter Conference on (pp. 464-472). IEEE.

[32] Smith, L. N., & Topin, N. (2017). Super-Convergence: Very Fast Training of Residual Networks Using Large Learning Rates. arXiv preprint arXiv:1708.07120.

[33] Howard, J., & Ruder, S. (2018). Fine-tuned Language Models for Text Classification. arXiv preprint arXiv:1801.06146.

[34] Pre-trained Models. (n.d.). Retrieved from https://github.com/tensorflow/models/tree/master/research/slim#pre-trained-models

[35] Source code for torch.nn.init. (n.d.) Retrieved from http://pytorch-zh.readthedocs.io/en/latest/_modules/torch/nn/init.html

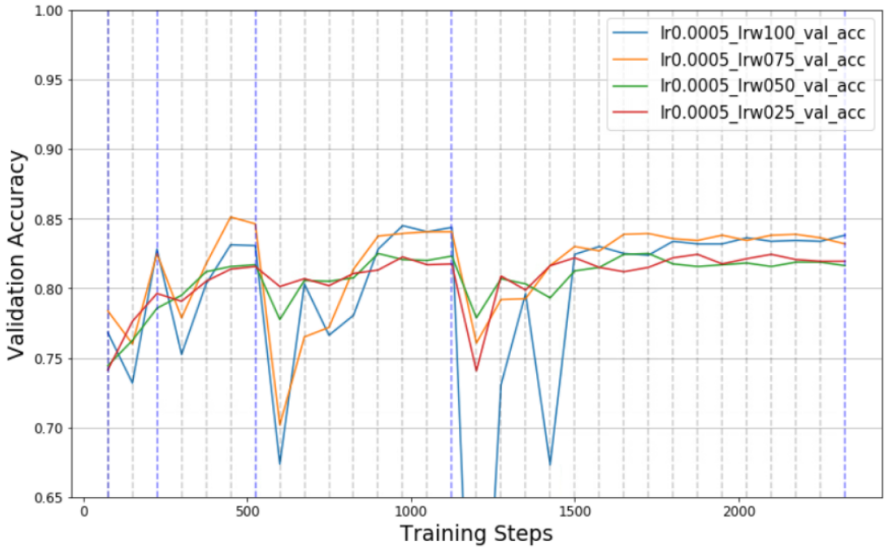
[36] Usage of initializers. (n.d.). Retrieved from https://keras.io/initializers/

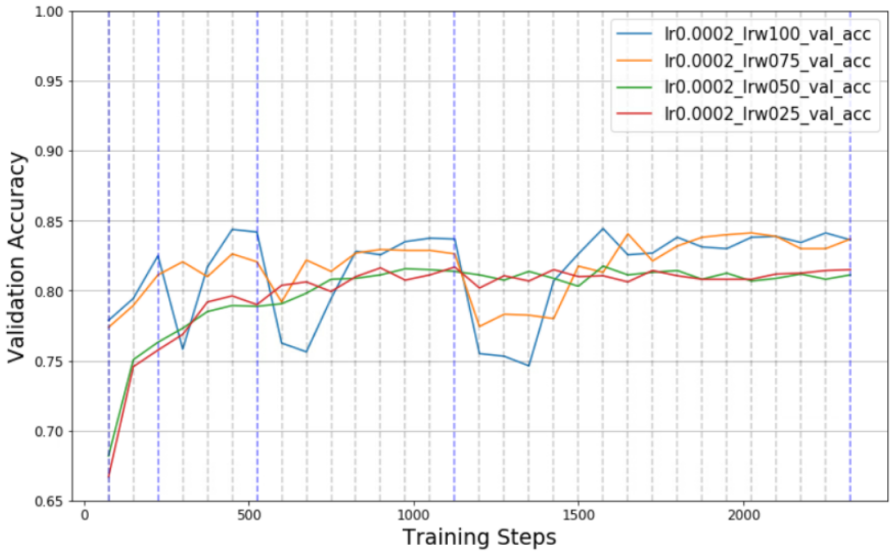
[37] Welcome to Kaggle Datasets. (n.d.). Retrieved from https://www.kaggle.com/datasets

[38] word2vec. (n.d.). Retrieved from https://code.google.com/archive/p/word2vec/

Appendix A

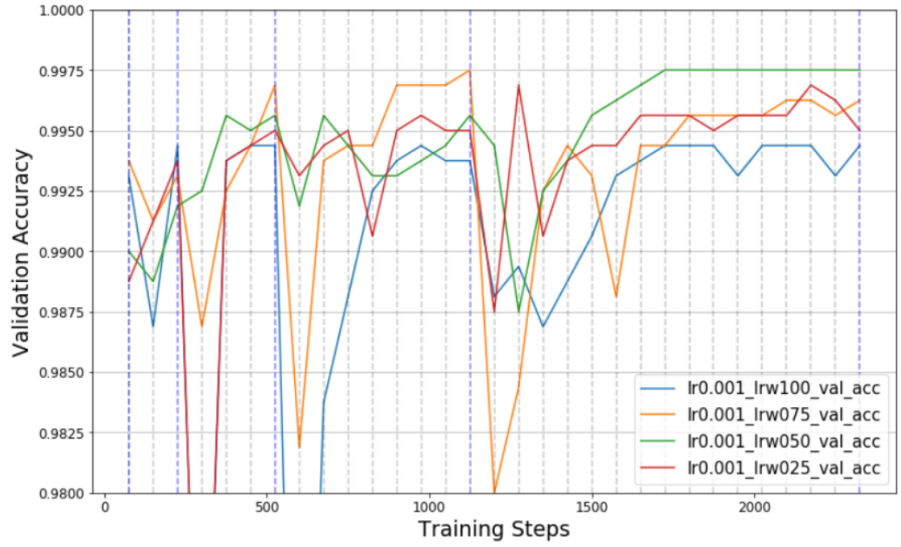
The model accuracy of Cifar01 to Cifar02 with a learning rate of 0.00005 and 0.00002, respectively.

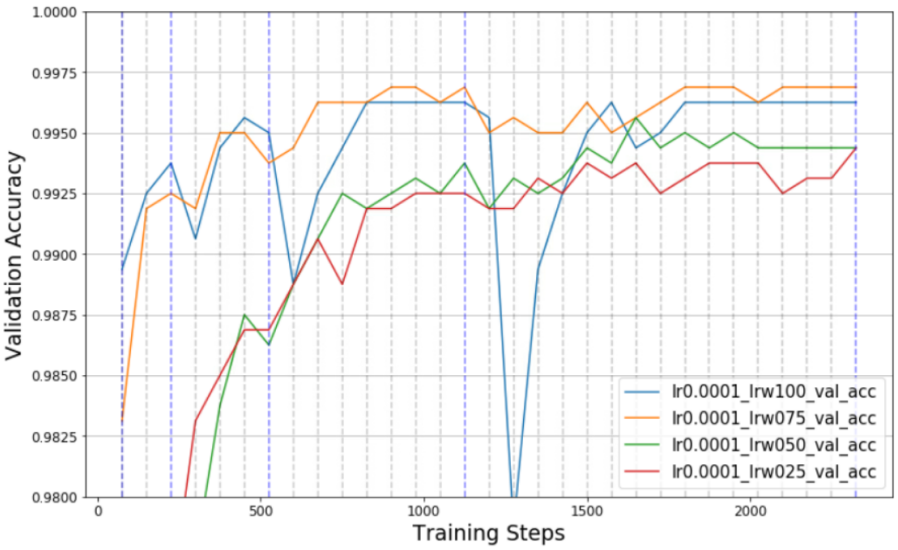




Appendix B

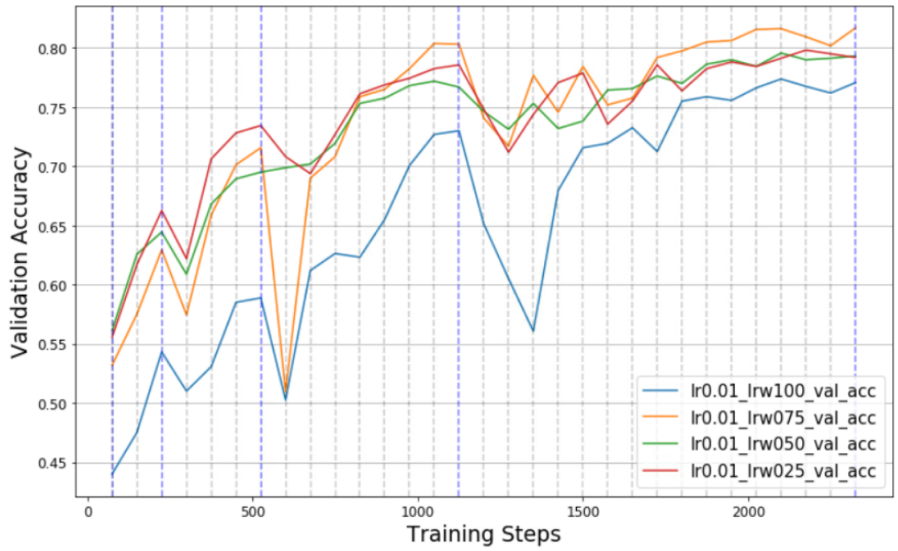
The model accuracy of Cifar01 to Emnist02 with a learning rate of 0.001 and 0.0001, respectively.

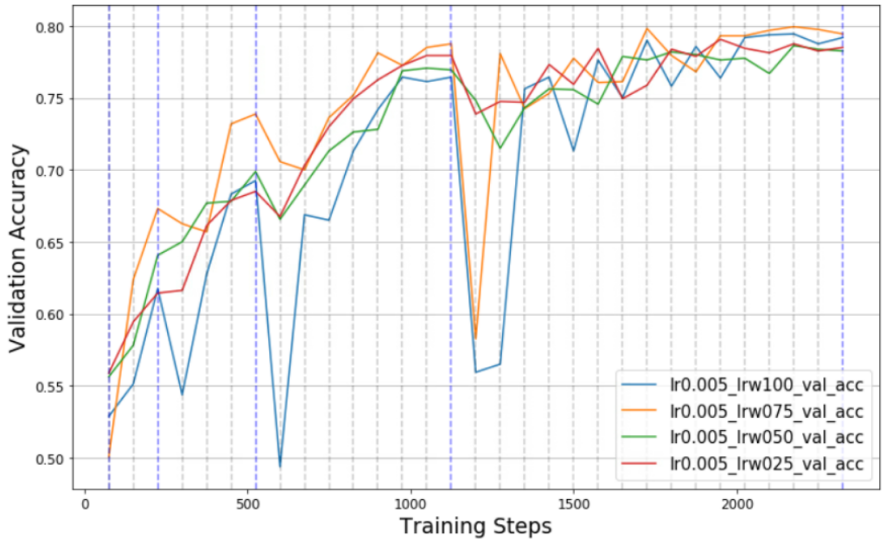




Appendix C

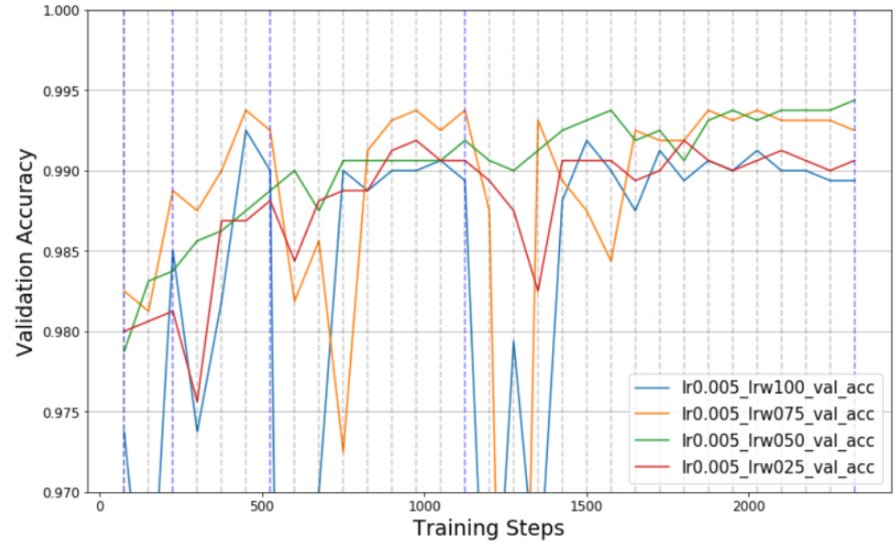
The model accuracy of Emnist01 to Cifar02 with a learning rate of 0.01 and 0.005, respectively.

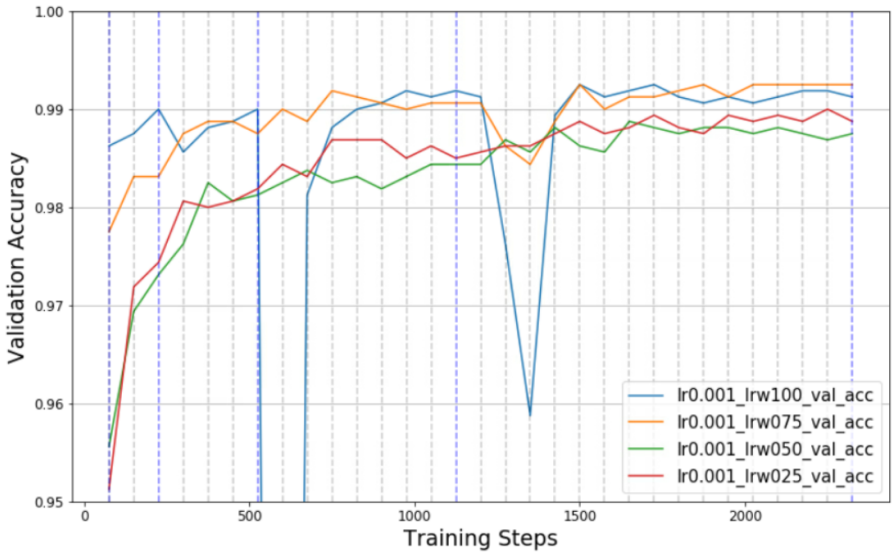




Appendix D

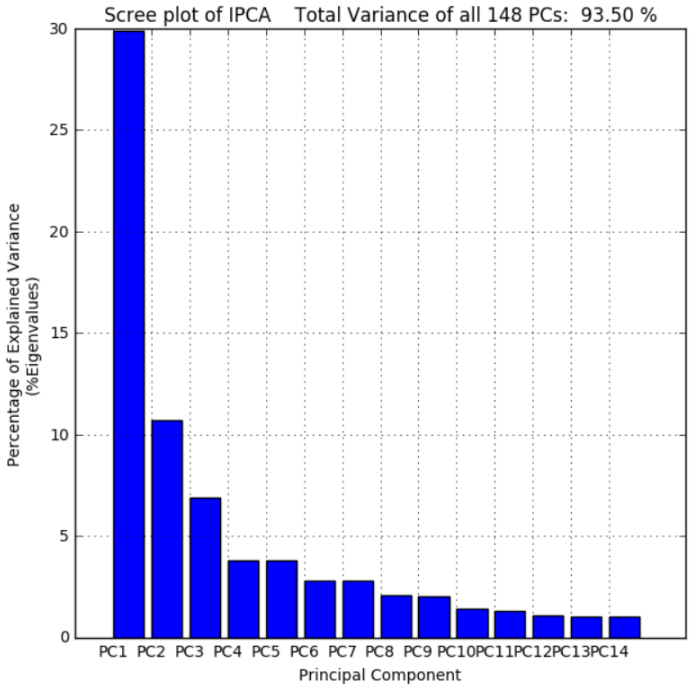
The model accuracy of Emnist01 to Emnist02 with a learning rate of 0.005 and 0.001, respectively.



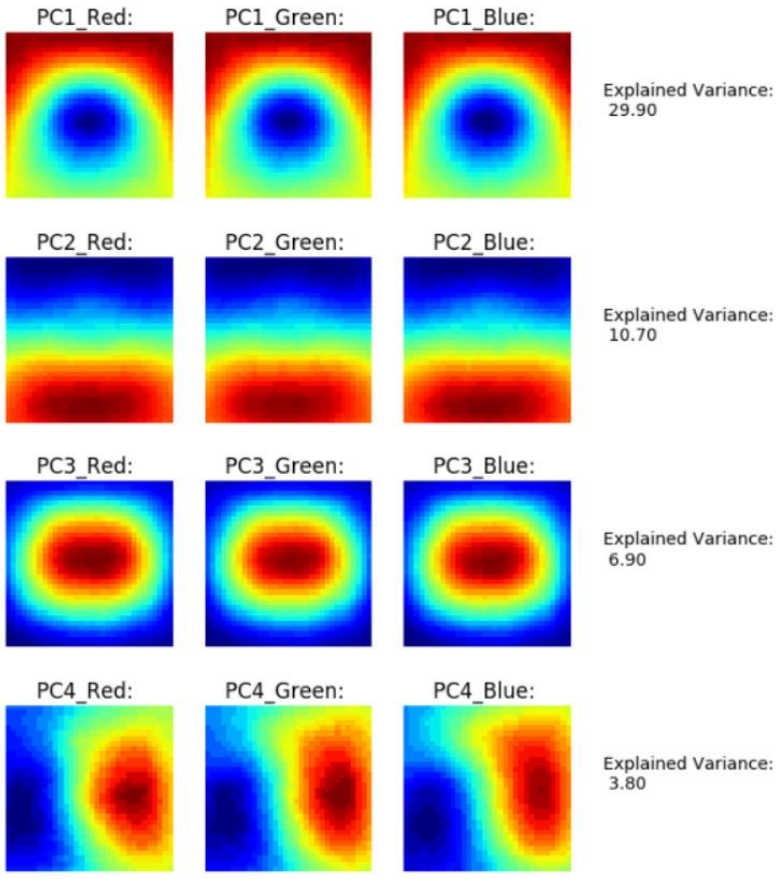


Appendix E

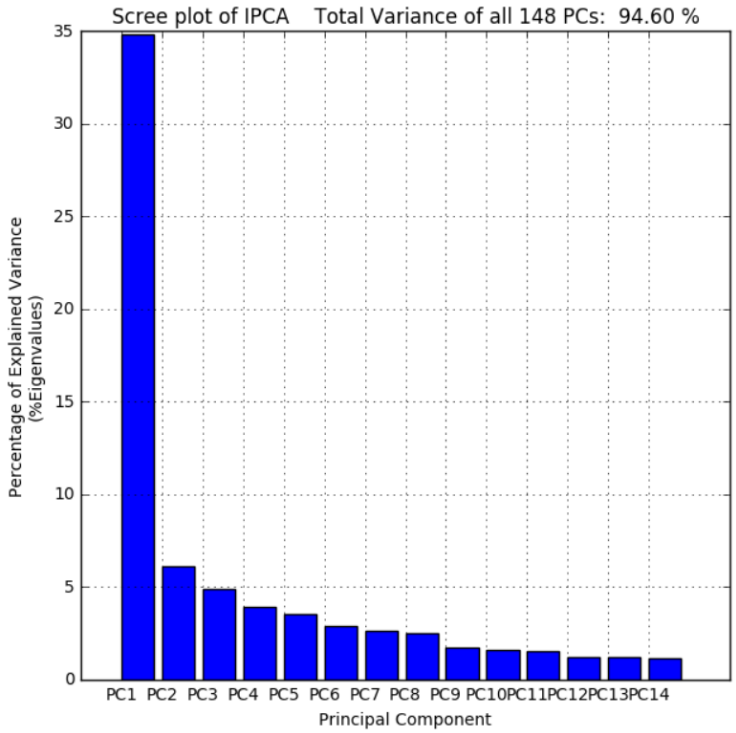
Appendix E1 - cifar01 to cifar02



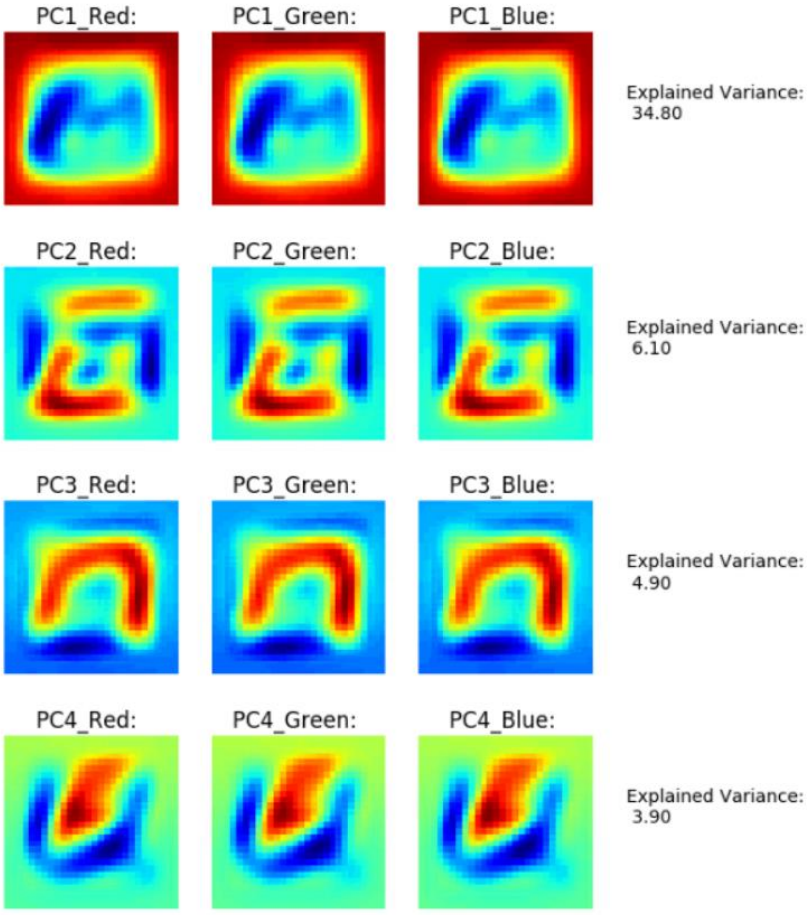
Four Most Dominant Principal Components:



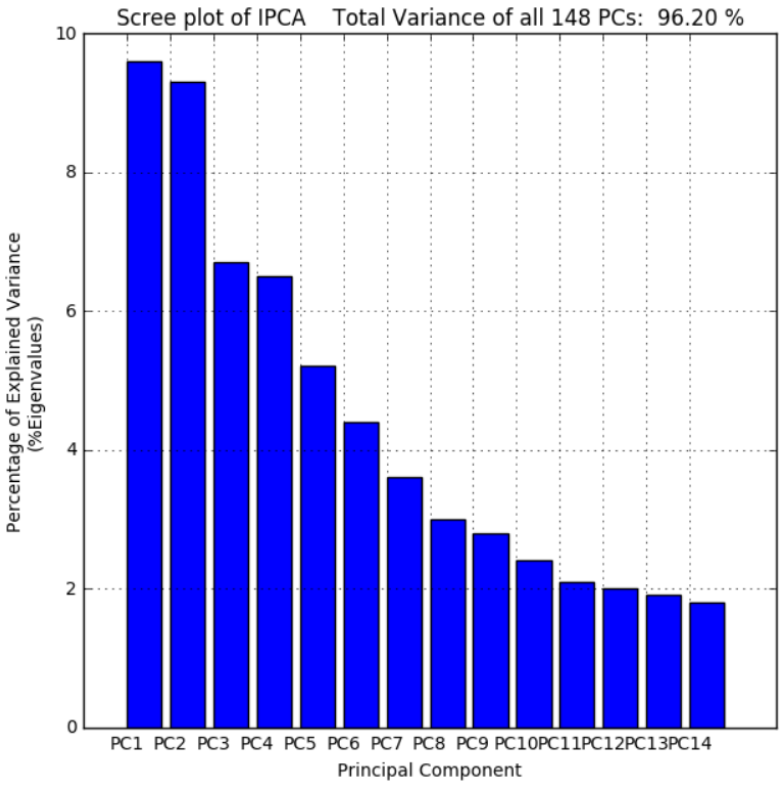
Appendix E2 - Emnist01 to cifar02



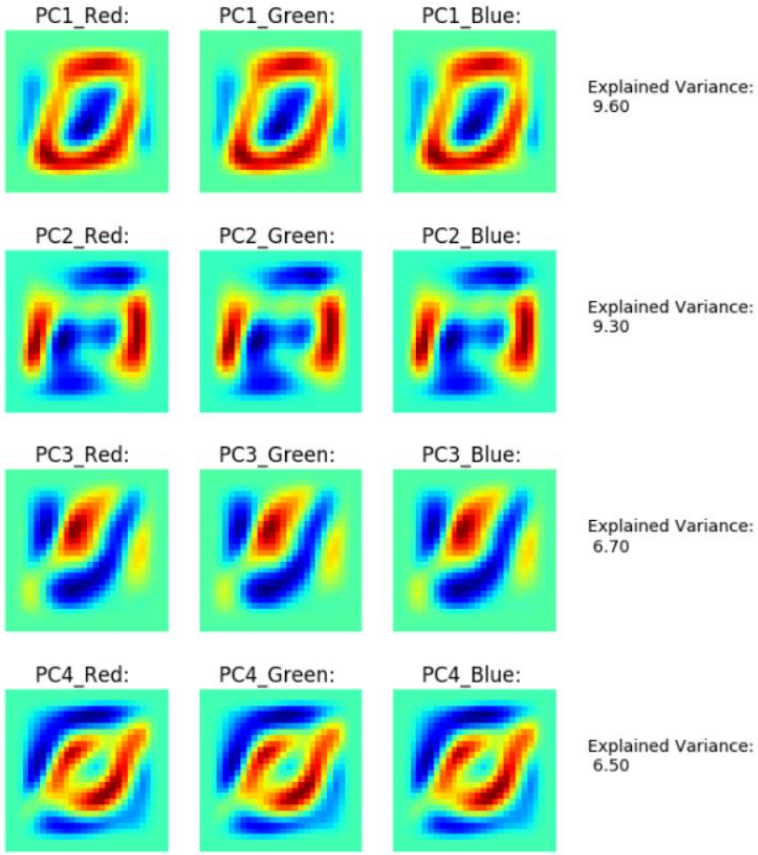
Four Most Dominant Principal Components:



Appendix E3 - Emnist01 to Emnist02

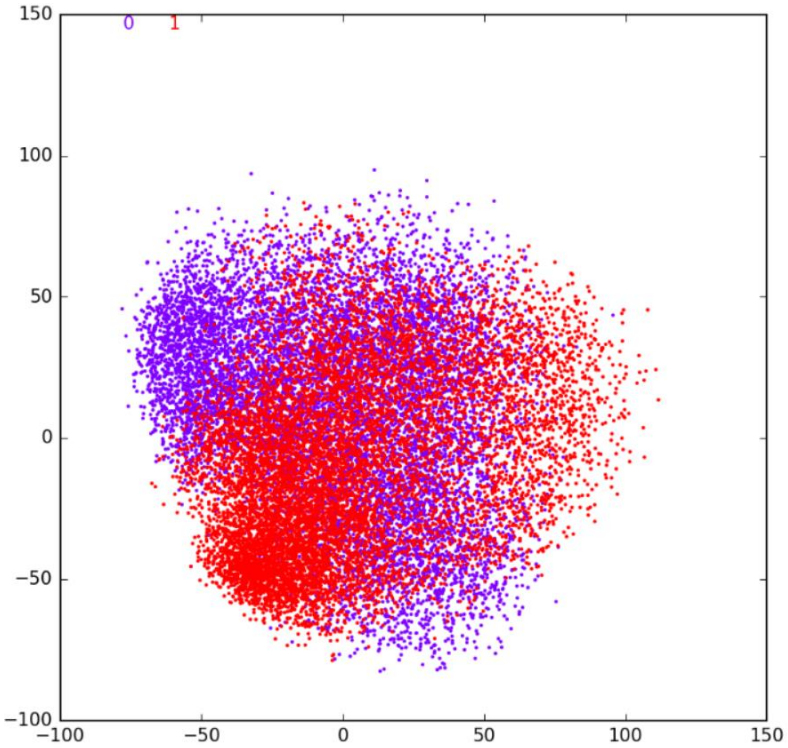


Four Most Dominant Principal Components:

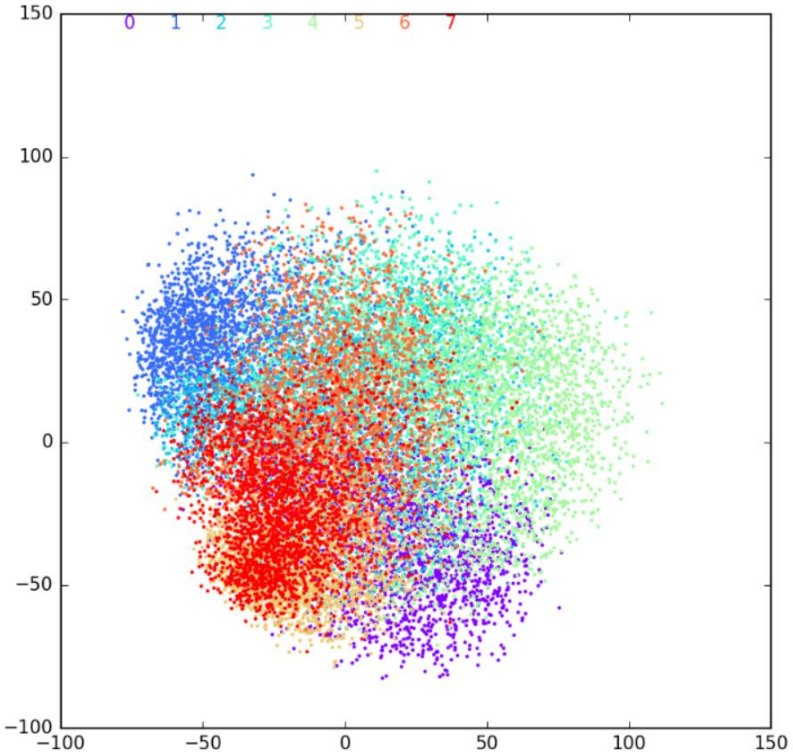


Appendix E3 - Emnist01 to Emnist02

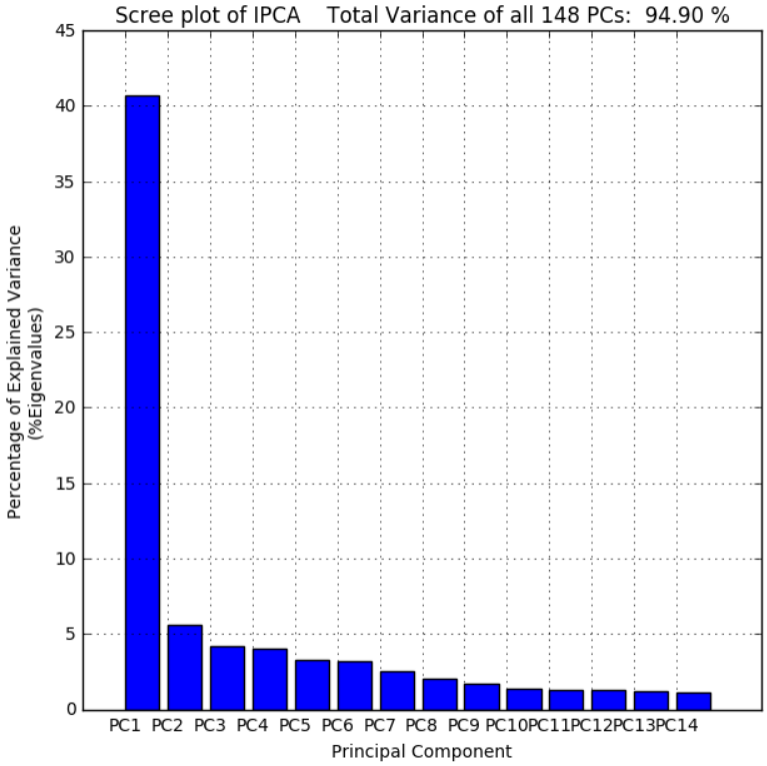
Scatter plot of the two most dominant principal components (red(1) = Em02, purple(0) = Em01):

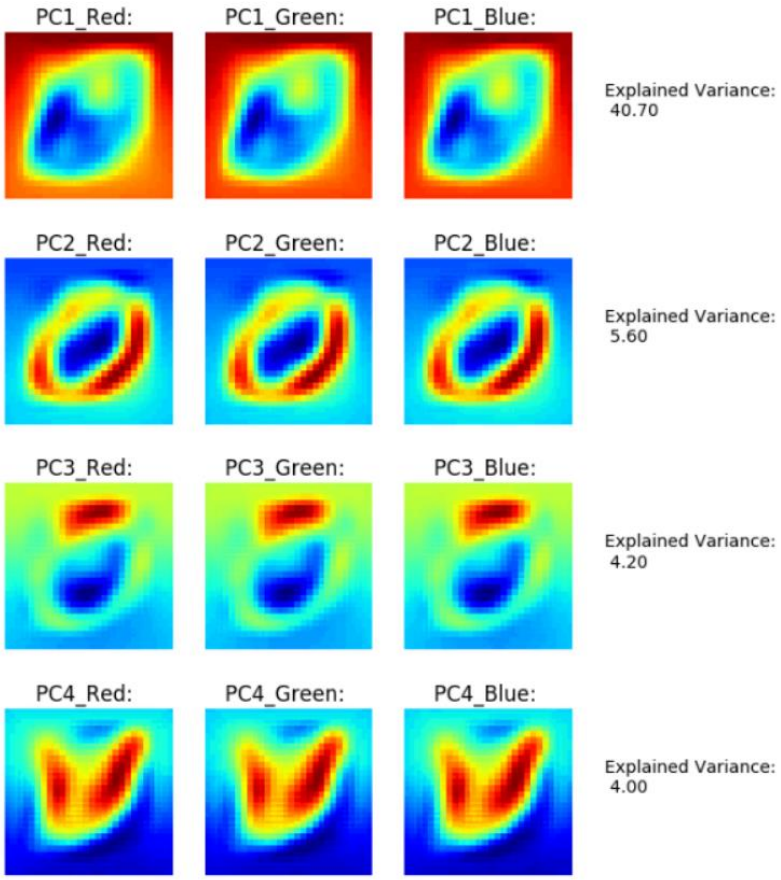


Scatter plot of the two most dominant principal components (4 to 7 = Em02, 0 to 3 = Em01):



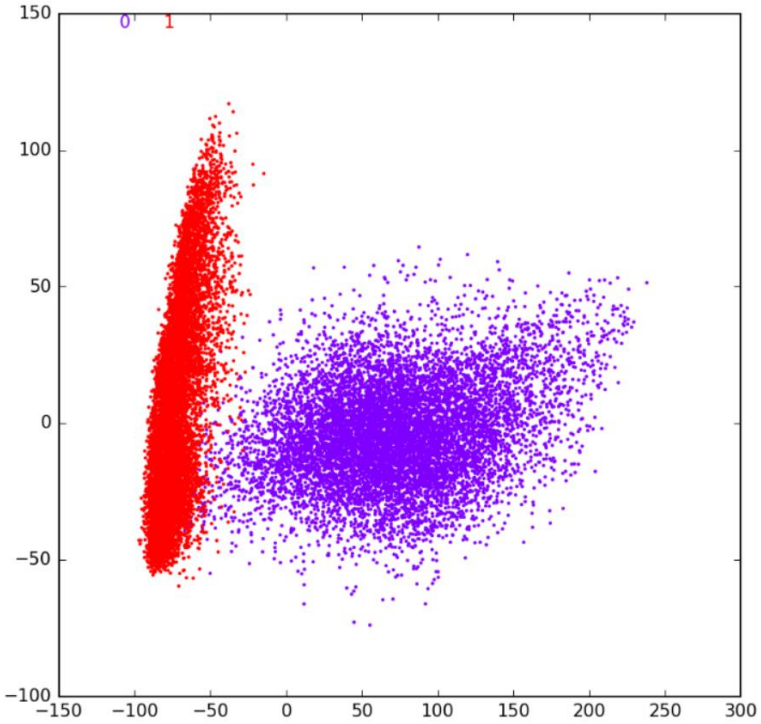
Appendix E4 - Cifar01 to Emnist02





Appendix E4 - Cifar01 to Emnist02:

Scatter plot of the two most dominant principal components (red(1) = Em02, purple(0) = Ci01):



Scatter plot of the two most dominant principal components (4 to 7 = Em02, 0 to 3 = Ci01):