Project 3

Emil Engelstad Moghaddam

December 16, 2020

Abstract

In the first part of the project, convolutional neural networks have been used to classify paintings and drawings from Pablo Picasso, Vincent van Gogh and Edgar Degas. The models showed that applying ReLU activation function in convolutional layers resulted in higher accuracy and faster optimization compared with the sigmoid activation function. The final optimised network model gave an accuracy of .83.

In the second part of the project, analysis of time series data has been performed to predict the future price of avocados. Neural networks using Long Short-Term Memory cells(LSTM), Gated Recurrent Units(GRUs), simple RNNs and a Convolutional LSTM were tested. The neural network using Gated Recurrent Units performed the best with a R-squared of .82 on test data.

1 Introduction

In 1997, the IBM Deep Blue supercomputer won over world chess champion Garry Kimovitsj Kasparov. Recently the Alphabet's DeepMind solved the protein folding mystery, and it seems there is nothing that AI cannot do. In this project, neural networks have faced the challenges of classifying art and predicting the future from past data.

Convolutional neural networks have had major success in classifying objects in images. However, identifying patterns in art is a task of different nature. Each of the three artists employ different motives for their paintings and drawings and each brings his own style. Their styles of course change during their lifetime, and the paintings are subject to changes in emotion and inspiration. The neural networks challenge is to find these underlying strings. Neural networks are reliant on having many training examples, to avoid overfitting. The dataset used in this project consists of only 2000 images, approximating to 670 Images from each artist.

Consistency in style and motive from a painter would be an advantage in training a neural network. For example, paintings from Edgar Degas often contain ballerinas, and the neural network would have an advantage of recognizing both his style and his preferred motive, much like a human professional would. As mentioned above, the challenge is that each of the 3 artists employ many different motives and styles. Recognizing style can be useful for style image searches and in the study of what makes art/pictures popular.

The second part of this project addresses the age old desire of humans to know or to predict the future. Humans have sought the wisdom of prophets, fortune tellers and shamans, who have looked into dreams, omens and the positioning of the stars to help fulfil their wishes. People have inquired about future love and tragedy, but also more practical matters like the next year's harvest. This obsession with the knowing future has not left humankind. Corporations often employ a suite of statistical tools to predict future price of stocks, or to gain a slight advantage over their competitors. Those who are able to predict the future correctly may turn a slight advantage into a major profit. Companies that could not, such as Blockbuster and Radioshack are left in the past.

Avocados are a great example of a commodity of which businesses may want to know future prices. The dataset contains information about prices and amount sold every week since 2015. New discoveries are being made every day in the field of artificial intelligence and many are wondering whether it can surpass other classical methods for future prediction.

1.1 Structure

The project starts with the theoreticalities of convolutional neural networks, then it covers the theoretical literature of Recurrent Neural Networks. Themes that were covered in project 1 and project 2, such as the MSE and L2 regularization have not been repeated.

There is one subsection with the implementation of the convolutional neural network, followed by one subsection describing the implementation of the RNN modifications employed on the time series data.

The results and discussion section is split in the same manner of fashion, with first a subsection of art classification followed by a subsection on time series prediction.

2 Theoretical model and technicalities

2.1 Convolution

In mathematics, convolution is an operation that takes two functions and produces a third function. Let's call the two functions f and g. One of the functions is reversed and shifted by a variable t. For example $g(\tau)$ becomes $g(t-\tau)$. Then the integral of the product f times g, is calculated for that specific t.
The convolution of f and g for some shift t can be written as in equation 1

$$(f * g)(t) = \int f(\tau)g(t-\tau)d\tau \tag{1}$$

In convolutional neural networks, a discrete two dimensional implementation of convolution is performed. This is because of the discrete nature of data in computers. One image on a computer is represented by three two dimensional matrices. The matrices represent the colours red, green and blue. Each spot in a matrix represents a pixel. The matrices hold values between 0 and 255 that determine the intensity of the designated colour for a specific pixel. The function g is replaced by a smaller matrix, usually 3 x 3 or 5x5, which contains some values, and is referred to by a filter or a kernel.

The kernel is placed so that it overlaps with some of the pixels of the image. The kernel is flipped, first horizontally then vertically, which is analogous to the reversal of the function g. Then the sum of the product between the value of each position in the kernel and pixel beneath is calculated. Computing sum of the products is the discrete implementation of the integral. The shift t is replaced with a vertical and a horizontal shift in the two dimensional convolution. Moving the positioning of the kernel on the picture is analogous to making a change in the shift t.

In figure 1 a 5x5 kernel is shifted both vertically and horizontally, compared with the top left corner which is often used as a t being equal to zero. The

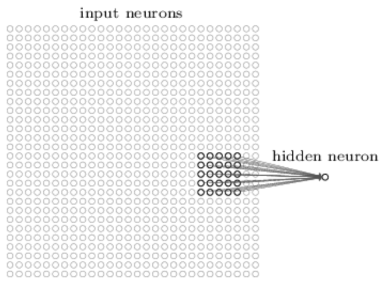

Figure 1: An illustration of discrete two dimensional convolution

darker circles represent the pixels which will be used in the calculation of the convolution.

$$y_{m,n} = \sum_j \sum_k w_{j,k} \times I_{m-j,n-k} \tag{2}$$

In equation 2 the shift t is replaced by a horizontal shift m and a vertical shift n. The integral is replaced with two capital sigma summation signs. The function $g(\tau)$ is replaced with the weights of the kernel, $w_{j,k}$. The sign for j and k in the image is positive while the sign for the weights in the kernel is negative which represents the reversal of the function g.

The convolution between a matrix and a kernel in a neural network, produces one output neuron for each possible positioning (or shift t), the kernel can be placed over the image.

All the hidden neurons share the same kernel, therefore they all detect the same feature, but at different locations of the image [1]. A feature may be anything that gets a kernel to produce a high value. The matrix beneath is an example of a kernel that will detect a vertical edge. If the matrix is placed on a uniform area of the image, the positive and negative values equal in magnitude and cancel each other. If, on the other hand, the kernel is placed at an area where the image goes from high intensities to low intensities, the output will be large.

$$g_{3,3} = \begin{pmatrix} a & 0 & -a \\ a & 0 & -a \\ a & 0 & -a \end{pmatrix}$$

In neural networks the kernel usually also contains a bias in addition to the weights. An additional parameter, usually called stride, determines how many pixels the kernel should move before calculating a new feature. The stride parameter determines the number of hidden neurons produced by the convolutional layer. There are two options for handling the edges of the image. One is placing the kernel so that the entire kernel is inside the image. The other is adding the padding with outside edges of the image. There are several options for padding. One can use zeroes, the mean value of the pixels, the mirror of the pixel values or circular padding where one imagines that the same picture is placed around the picture.

When convolution is done without the reversal of one of the functions, the mathematical operation is then called cross-correlation. Convolution is commutative and when using the Fourier transformation convolution becomes addition in the frequency domain. These properties do not hold for cross-correlation. However, in neural networks the properties of convolution are often lost due to the use of non linear operations such as activation functions. Therefore convolution is often substituted for cross-correlation in convolutional neural networks.

2.2 Activation function

The values received at the hidden neurons are usually transferred through an activation function. The Rectified Linear Unit function, $R(z) = max(0, z)$, is the most commonly used activation function in modern convolutional neural networks. Other functions such as the sigmoid and the tan hyperbolic, which are common in multi layer perceptrons, can also be used in convolution. The derivative for the ReLU function is when z is larger than zero, is equal to one. The derivative of the sigmoid on the other hand has a maximum value of 0.25. Neural networks with many layers often suffer from what is called the vanishing gradient problem, the gradient becomes smaller and smaller for each layer in the backpropagation algorithm. The ReLU solves this problem by propagating the entire gradient. The Relu is fast to calculate as the derivative is known, and only requires a comparison. The Sigmoid has the advantage of being non linear and having outputs in a limited range. The gradient algorithm however is decreasingly effective when the sigmoid receives values outside a limited range around zero [2].

2.3 Pooling layer

One often uses a pooling layer after a convolution layer. This pooling layer is used to simplify the information from the convolutional layer. Pooling layers take a region of the transformed convolutional output, usually 2x2, and use it to compute one single value from that region [1] . Two commonly used techniques are max pooling and average pooling. The former works by taking the max value of the region. Average pooling works by taking the average value of the region. A 2x2 hidden region of nodes will contain information connected to a slightly larger region of the image. Max pooling will give the information about whether a feature exists in that slightly larger region of the image, while average pooling gives information about how strongly the feature is detected on average in that area. The intuition is that the exact position of the feature isn't so important to a network.

2.4 Overfitting

Neural networks are heavily reliant on having many training examples to avoid overfitting. Overfitting is when a network learns a function with high variance and low bias to perfectly fit the training data. Due to sampling noise, neural networks may find relationships that only exist in the training dataset and not the test set, even if they are drawn form the same distribution. [3] Four commonly used methods of reducing overfitting is data augmentation, batch normalization, dropout and transfer learning.

2.4.1 Data augmentation

Data augmentation is a technique used to increase the data quantity by adding slightly modified copies of already existing data to the data set. In the case of convolutional networks, this can be done by slightly moving the image, zooming in, rotating, shearing , flipping, noise injection etc. Data augmentation acts as a regulator that helps reduce overfitting. [4]

2.4.2 Batch normalization

Batch normalization is a method proposed by Sergey Ioffe and Christian Szegedy in 2015 to reduce internal covariate shift. [5]

This is a phenomenon that occurs during training of a neural network where the distribution of the inputs received at each layer changes, because of the weights of the network change. ICT slows down training efficiency by requiring a lower learning rate. When the neural network uses activation functions with saturating nonlinearities, such as the sigmoid and hyperbolic tangent, this problem is extra prevalent. [5]

A later paper argues that the effectiveness of batch normalization does not come from reducing the internal covariate shift [6]

Batch normalization is performed by calculating the mean and the variance of each mini batch at specific layers of the network. The mean and variance is used to transform the values so that they have mean close to zero and variance close to one. The normalization is done on a per feature basis. The use of minibatches is Stochastic gradient descent is based on the assumption that the batch gradient is a good approximation of the true gradient. A similar assumption is made in batch normalization where one assumes that the distribution of the minibach is close to the true distribution. After normalization, the values are scaled and shifted, as shown in equation 3, by parameters γ and $beta$, which are learned by the network. [5]

$$y^{(k)} = \gamma^{(k)}\bar{x} + \beta^{(k)} \tag{3}$$

2.4.3 Dropout

Dilution (also called Dropout) is a regularization technique for reducing overfitting in artificial neural networks. The key idea is to randomly drop units, along with their connections, from the neural network during training. [3] . One possible implementation of dropout is having each unit retained with a probability p independent of other units. An equivalent way of viewing dropout is that one is sampling a thinned network from a larger set. If a unit is present with probability p during training, the outgoing weights of that unit are scaled down by p at test time. [3]

Dropout works by preventing neurons to evolve complex co-adaptations with other hidden units. When dropout is applied each hidden neuron in a neural network must learn to work with a random set of other neurons. This should help the hidden neuron to evolve helpful features on its own, and not rely on other units to correct its mistakes. [3]

2.4.4 Transfer learning

Transfer learning offers the chance for CNNs to learn with limited data samples by transferring knowledge from models pretrained on large datasets. [7]

2.5 Recurrent Neural Networks

Recurrent neural networks are often used to predict the future from time series data. Recurrent neural networks look very much like feed forward neural networks, with the exception that they also have layers with connections pointing backwards. These layers are called recurrent layers. Just like a normal layer of a FFN, a recurrent layer receives an input vector x. The input vector x is either from the input network or the output from a previous layer. Through the belonging weights and biases of the recurrent layer, the layer produces an output h. As we continue to feed data points to the network, the recurrent layer will receive input vectors $x_{t-2}, x_{t-1}, x_t, x_{t+1}, x_{t+2}$. For what we call each timestep t the recurrent layer will produce output $h_{t-2}, h_{t-1}, h_t, h_{t+1}, h_{t+2}$. What separates the recurrent layer from a standard multi-layer perceptron layer, is that for timestep t the recurrent layer is also using its own output from the previous timestep h_{t-1} as input, as an addition to the input x_t. If we use w_x to denote the weights for input x_t and w_h to denote the weights for previous output h_{t-1}, equation 4 can be used to describe the output of the recurrent layer. [8]. ϕ is the activation function and b is the bias.

$$y_t = \phi(x_{(t)}^T \cdot w_x + h_{(t-1)}^T \cdot w_y + b) \tag{4}$$

A recurrent neural network can be viewed as a feed forward neural network with a loop in the hidden layer. The loop acts as a sort of working memory for the recurrent neural network. The unrolling of the loop lets us view the RNN like a chained series of feed forward neural networks, all sharing the same structure, weights, biases and activation function, with an additional working memory. Figure 2 illustrates these two equivalent ways of viewing the network.

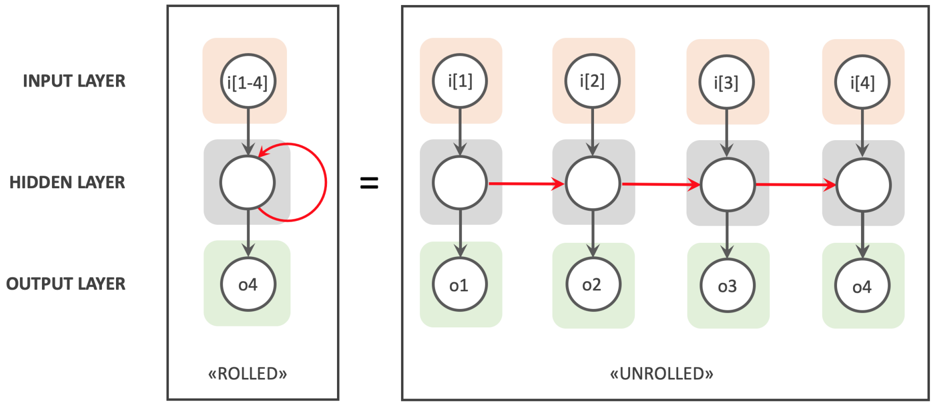

Figure 2: Unrolling a Recurrent Neural Network

2.6 Back-propagation through time

For optimization of a recurrent neural network, the normal back propagation algorithm is replaced with a back propagation algorithm on the unrolled network. This is known as back-propagation through time, or BPTT. When we unroll the network, each timestep will have one input, one copy of the network and one output.

With time series data, we start the algorithm by first giving a sequence of input and desired output pairs to the RNN. This could be the price of a particular stock for each of m days before a given day as input, and the price of that stock for the next day as the desired prediction.

We call the model's prediction for each timestep of the unrolled sequence as $\tilde{y}_i$, where i indicates the position in the sequence. The loss for each timestep will be given by the prediction and the actual value y_i. The loss of timestep i can be written as $L_i(\tilde{y}, y)$.

Using the chain rule we can calculate the derivative of the loss function for timestep i with respect to the weights between the hidden layer and the output layer. We can also calculate the derivative of the loss function for the weight matrix with respect to the input layer. To calculate the derivative of the loss with respect to w_h, we must follow the unrolled loop backwards. This is because the output of layer i depends on the values at each hidden layer before through w_h. The value at a received at hidden node number 4 can be written like this $w_h(w_h(w_h(h0 + w_x * i_1) + w_x * i_2) + w_x * i_3)$. Here we see that there are many derivatives to be calculated for the gradient.

Earlier we explained that deep architectures can have problems caused by the vanishing of gradients. This problem is also common in recurrent neural networks, but the vanishing of the gradient occurs with back-propagation through the timesteps. From linear algebra, we know that singular values represent the expansion or shrinkage of the accompanying vector. The largest singular value shows the maximum possible extension of a vector by a matrix. If the largest singular value is smaller than one, the gradient will vanish.If the largest singular value is bigger than one, the gradient will expand [9]. We also iterate through the derivative of many activation functions, which might diminish the magnitude of the gradient depending on the chosen function. [10]. Large gradients can be handled by shrinking the gradient, if it is larger than some threshold. Vanishing gradients are often handled by changing the architecture to a LSTM.

2.7 LSTM

The Long Short-Term Memory (LSTM) cell was proposed in 1997 by Sepp Hochreiter and Jorgen Schmidhuber [8]. Figure 3 illustrates the LSTM. Just like an RNN a LSTM passes on its hidden state. We have seen that hidden states carry information from previous states, but because of the vanishing gradient problem it is bad at carrying information from long back. The hidden state acts like the short term memory for the LSTM. The major difference between the standard RNN and the LSTM is that the LSTM also contains a state vector with important information from the past.

LSTM cells have gates that determine what information should be **forgotten**, what additional **input** should be added, and what should be the **output** to the dense layer. The gates create floating values between one and zero for each index of the state vector. These values are created using the sigmoid function.

Weight matrices W_f, W_i, W_o are used to create three filters, based on the current input x_t and the last hidden state h_{t-1}.
$f_t = \sigma(W_f \times [h_{t-1}, x_t] + b_f)$
$i_t = \sigma(W_i \times [h_{t-1}, x_t] + b_i)$
$o_t = \sigma(W_o \times [h_{t-1}, x_t] + b_o)$

The forget filter is used directly on the state vector $C_t = C_{t-1} * f_t$, by pointwise multiplication. Just like in a RNN, the input and hidden state is transformed by a weight matrix and the hyperbolic tangent activation function (tanh). The input filter is used on these transformed values to determine what is added to the state vector. $C_t += tanh(W \times [h_{t-1}, x_t] + b) * i_t$. The output filter is used on the tanh transformed state vector to produce the output. $h_t = tanh(C_t) * o_t$.

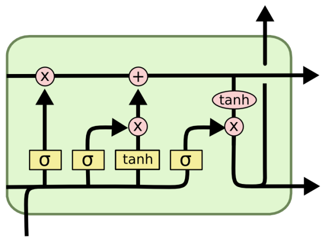

Figure 3: Long Short-Term Memory cell

2.8 Gated Recurrent Unit

The Gated Recurrent Units(GRUs) are gating mechanisms in recurrent neural networks, first introduced in 2014 by Kyunghyun Cho et al [11] . GRUs act quite like LSTM cells, with the exception of not having a cell state.

The GRU produces one update vector z and one reset vector r, which act like the gates in the lstm.
$z_t = \sigma(W_z[h_{t-1}, x_t] + b_z)$
$r_t = \sigma(W_r[h_{t-1}, x_t] + b_r)$
These gates are used on the precious hidden states and new input data to produce the new results, like described in the formula below :
$h_t = h_{t-1} * z_t + (1 - z_t) * tanh(W[x_t, r_t * h_{t-1}] + b)$

2.9 Implementation - Convolutional neural network

In this project two different convolutional neural network architectures have been tested. The first neural network consists of one convolutional layer with 128 kernels, a 2x2 pooling layer and a output layer with the cross entropy as the loss function. The kernel size used is 5x5. All the images were resized to 224x224 pixels, and both horizontal and vertical flipping has been used for data augmentation.

For this neural network we have compared the sigmoid activation function with the ReLU activation function. We have also compared max pooling with average pooling.

The second convolutional network has also started with a convolutional layer with 128 kernels and a 2x2 pooling layer. Then a second convolutional layer with 64 kernels and another 2x2 pooling layer. Then there is one fully connected layer of 75 nodes before the final output layer.

For the second convolutional layer, we have again compared the sigmoid and ReLu as activation functions. We also trained the network with different varying implementations of data augmentation. The effect of dropout and batch normalization was determined. A larger 10x10 kernel was also tested.

A convolutional neural network has been restricted to use only one convolution and the images produced were saved.

The Adam optimizer was used.

Transfer learning was tested on Inceptionv3. Inceptionv3 is the third edition of Google's Inception Convolutional Neural Network, originally introduced during the ImageNet Recognition Challenge.

2.10 Implementation - Recurrent neural networks on time series

Four different implementations of recurrent neural networks have been tested on the avocado dataset.

The first implementation contained LSTM cells, the second implementation used GRU's, the third implementation used the standard simple RNN' cells, and the fourth implementation employed both one dimensional discrete convolution and LSTM cells.

The three first implementations used four layers of 50 units each, of the designated cell type. The fourth implementation used 128 convolutions followed by a pooling layer of size 2 , then another 128 convolutions followed by another equal pooling layer. Two layers with LSTM cells with 60 units each. A dropout of .2 was used in all four models.

The models have been used to predict the future price of avocados. Three different tests have been used. In the first test the models used the price of avocados on 60 previous timesteps. In the second test the models used the amount sold on 60 previous timesteps. The third test the models used the price of avocados for the 30 previous timesteps, and the average amount sold in 30 previous timesteps.

The RMSprop optimizer with a default learning rate of 0.001 was used.

3 Results and discussion

3.1 Art classification

Table 1: Convolutional models for Art Classification

Conv. layers	Hidden units	Activation Function	Pooling method	Epochs	Test accuracy
One	0	Sigmoid	Max	170	.65
One	0	ReLU	Average	20	.76
One	0	ReLU	Max	25	.75
Two	75	Sigmoid	Max	175	.70
Two	75	ReLU	Max	45	.83

Tabel 1 shows how different models performed on categorizing art from Pablo Picasso, Vincent van Gogh and Edgar Degas. The results are read from graphs and therefore prone to minor errors. The column heading "Epochs" shows how many epochs are needed to reach the accompanying test accuracy. The results are based upon the assumption of early stopping of the model in an optimal region of training time.

Table 1 shows that models employing the ReLU reach better results and are significantly faster to train than models using the Sigmoid. The deeper models also outperformed the shallow models. The rest of the results are modifications of the last model.

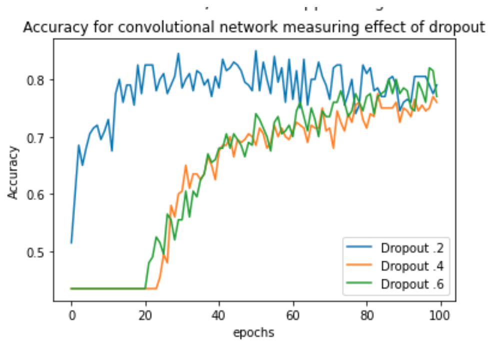

Figure 4: The effects on dropout on test accuracy

Figure 4 shows the effects of dropout on the test accuracy of the convolutional neural network. In the implementation the lambda regularization parameter has been reduced from 0.00006 to 0.00003. The reason being that results indicated that the network was already highly regularized. The figure indicates that an additional dropout of .2 might be beneficial, but the graph is not very clear.

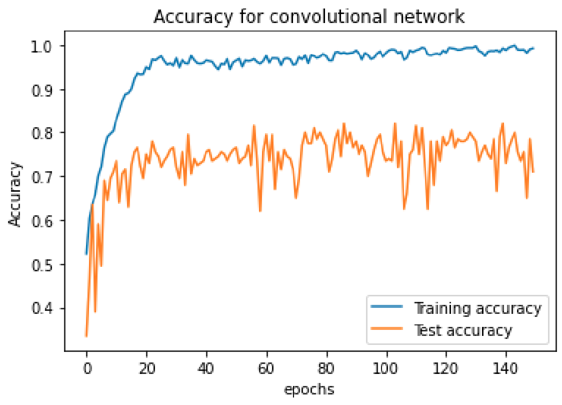

Figure 5: The effects of batch normalization on test accuracy

Figure 5 shows that batch normalization did not further improve the results of the convolutional neural network. Here also the lambda regularization parameter has been reduced from 0.00006 to 0.00003. According to the theory on internal covariate shift batch would have been more effective if the activation function was sigmoid or the hyperbolic tangent

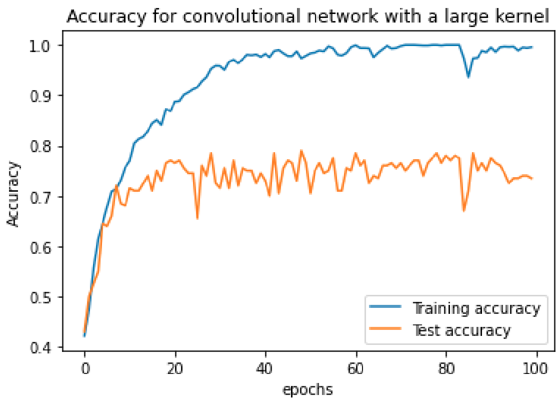

Figure 6: The effects of a 10x10 kernel on test accuracy

Figure 6 shows that a larger kernel size decreased the performance of the model.

Figure 7 shows that a horizontal and vertical flipping outperformed no data augmentation. The figure also indicate that adding extra rotation, shear and zooming(the green line) dosent improve the results. However the graph ends

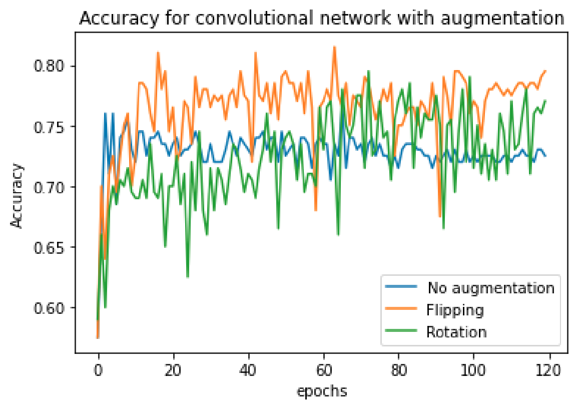

Figure 7: The effects of data augmentation

with the green line beeing on a upward trend. Therefor this might be the wrong conclusion. From frigure 7 we can aslo se that data augmentaion slows the training proseses.

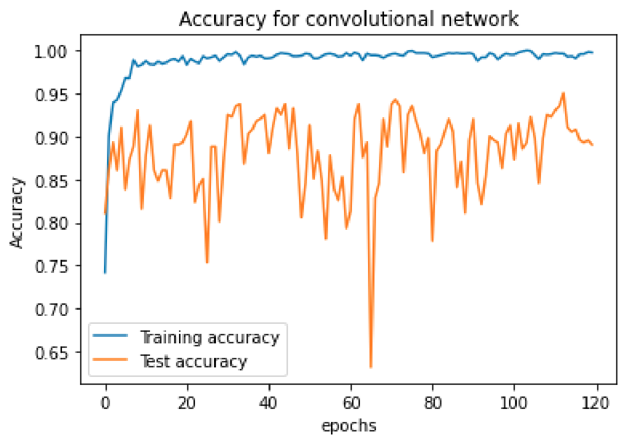

Figure 8: Test and training accuracy for Inception V3

Figure 8 shows the results form inception V3. The results are in a range of .80 to .95.

Figure 9 10 11 12, are the result of a neural network being constricted to only use convolution of size 5x5. The one convolution corresponds to three matrices, one for each color.

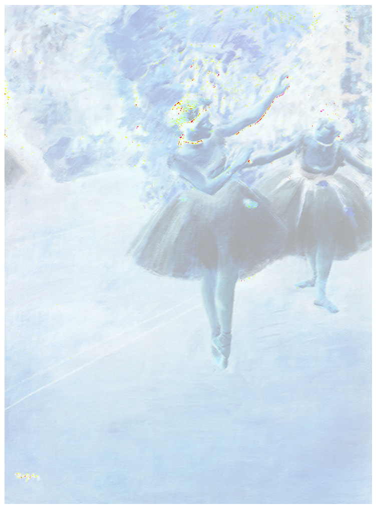

Figure 9: Edgar Degas

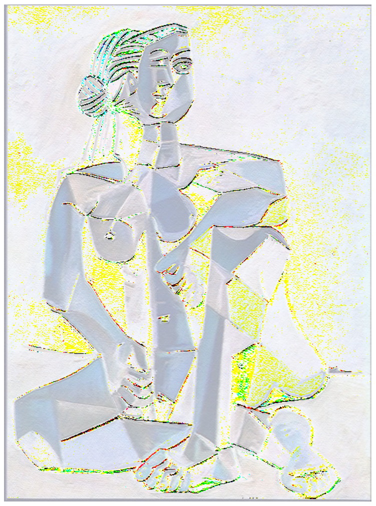

Figure 10: Pablo Picasso

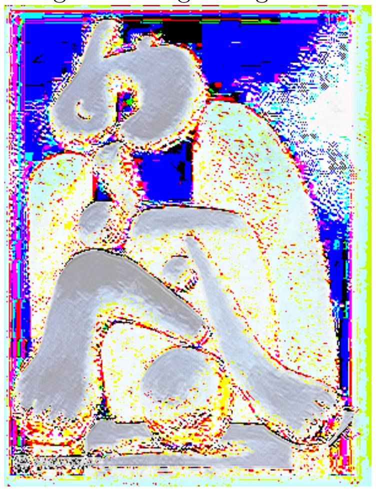

Figure 11: Pablo Picasso

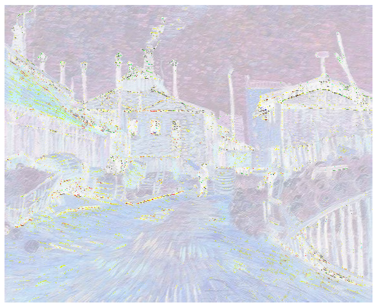

Figure 12: Vincent van Gogh

3.2 Time Series data

Table 2: Recurrent Neural networks - MSE test

Type	Price	Amount	Price + Amount
LSTM	0.00466	0.08552	0.01395
GRU	0.00426	0.0163	0.00584
Simple RNN	0.00428	0.05002	0.0062
Convolutional LSTM	0.00879	0.04746	0.05454

Tabel 2 shows the four different models compared on the three tests. Volume of avocados sold is negatively correlated with r = -.42 to the average price of avocados. Therefore the models using price plus amount sold should in theory be better able to predict the future price of avocados. The table shows that they performed worse. However the models could not separate the noise from the true pattern and adding information about volume sold reduced the results. This LSTM and Convolutional LSTM performed much worse when information about volume sold was added. The neural network with Gated Recurrent Units outperformed the other methods on all tasks. The results suggest GRUs are better at handling noise. These results indicate that additional feature information should be structured in a way to help improve predictions. The volume sold would likely be a better predictive variable if the average price in a country was scaled compared to the amount sold.

However the models could not separate the noise from the true pattern and adding information about volume sold reduced the results. This LSTM and Convolutional LSTM performed much worse when information about volume sold was added. The neural network with Gated Recurrent Units outperformed the other methods on all tasks. The results suggest GRUs are better at handling noise. These results indicate that additional feature information should be structured in a way to help improve predictions.

An additional test where the four layer GRU model was trained with the 60 previous timesteps of avocado prices and 60 previous timesteps of average amount sold showed no improvements, and did not explain why the "price + amount" models don't outperform the only "price models".

The models trained on previous prices showed significant differences in which epoch the validation data reached its minimum. The convolutional lstm reached its validation minimum after 87 epochs, the four layer lstm reached its minimum after 137 epochs, the Gated Recurrent Unit reached its validation minimum after 426 epochs and the simple RNN reached its validation minimum after 688 epochs.

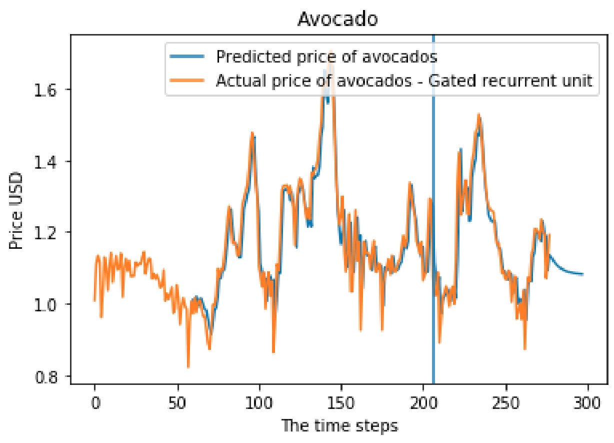

Figure 13: Time series prediction with Gated Recurrent Units using Price

Figure 13 displays how the four layer GRU was able to predict the future price of avocados. The entire prediction has a R - squared of .85 while the test plus validation part has a R - squared of .82. The end of the blue line shows how the model would predict 20 weeks into the future. The time axis shows weeks since 4. January 2015.

4 Conclusions

In conclusion, I have developed 2 neural networks for 2 data sets with very different characteristics.
The first involved training convolutional neural networks to identify paintings by Pablo Picasso, Vinvent van Gogh and Edgar Degas from one another. Here the aim was not to identify the painting but rather the artist from the style of the painting. Here we saw that using a deeper architecture and switching to ReLU activation improved the results. This network achieved an accuracy .83 despite a limited training data set size. Further regularization through batch normalization and dropout, on top of L2 norm regularization and data augmentation, did not improve the final model. The results indicate that using one regularization technique limits the effectiveness of others.

The second data set was a time series of avocado prices. Neural networks using simple RNN's, Gated Recurrent Unit's and LSTM's all performed well in predicting the future price of avocados based on previous prices. However, the models could not predict prices accurately from volumes of avocados sold from earlier time points, where correlation was 0.42. Including the volume of avocados sold as a feature did not improve accuracy of the model. The final model, using GRU's, gave an R-squared of .82 for predicting prices of avocados by one week forward in time on the last 72 weeks of the dataset.

4.1 Recommended future work on the topic

For the art categorization it would be interesting to see whether the neural networks could predict the popularity of the artwork made.

For the time series dataset it would be interesting to see how external factors, such as inflation rate, crop yield and currency fluctuations of producer countries, could be used to explain and reduce the residuals.

References

[1] M. A. Nielsen, *Neural Networks and Deep Learning.* Determination Press, 2015.

[2] G. Montavon, G. Orr, and K.-R. Mller, *Neural Networks: Tricks of the Trade.* Springer Publishing Company, Incorporated, 2nd ed., 2012.

[3] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, pp. 1929–1958, 2014.

[4] T. M. Shorten, Connor; Khoshgoftaar, *A survey on Image Data Augmentation for Deep Learning.* Springer Publishing Company, Incorporated, 2015.

[5] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *CoRR*, 2015.

[6] S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry, "How does batch normalization help optimization?," 2019.

[7] K. Wang, X. Gao, Y. Zhao, X. Li, D. Dou, and C.-Z. Xu, "Pay attention to features, transfer learn faster cnns," 2020.

[8] A. Gron, *Hands-On Machine Learning with Scikit-Learn, Keras, and Tensorflow.* O'Reilly Media, 2017.

[9] F.-F. L. . J. J. . S. Yeung, "Recurrent neural networks lecture 10," May 4, 2017.

[10] A. Soleimany, "Mit introduction to deep learning 6.s191: Lecture 2," January 2020.

[11] K. Cho, B. van Merrienboer, Ç. Gülçehre, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," *CoRR*, 2014.