Image Classification Using Different Neural Networks

†Nishanta Baral
†Montclair State University, Montclair, New Jersey, 07043

***Abstract*—Machine learning algorithms and neural networks are widely used in the field of image recognition and image classification. The kernel of neural networks is to optimize over a composition function that is solved using gradient descent and back propagation. In this paper, we take a dataset that contains 25K pictures of dogs and cats, and investigate different neural networks and machine learning algorithms for image classification. We firstly investigate with a simple Artifical Neural Network (ANN) and develop a more sophisticated Convolutional Neural Network (CNN) for image classification. Similarly, we use Principal Component Analysis (PCA) that focuses on dimensional reduction to train a machine learning algorithm.**

***Index Terms*—Artificial Neural Network (ANN), Machine Learning (ML), Neural Network, Convolutional Neural Network(CNN), Principal Component Analysis (PCA).**

I. Introduction

The human brain is said to be the the grandest biological frontier and the most complex thing we have yet discovered in our universe. [1] The capacity of the brain is massively huge with about 2.5 million gigabytes memory space. Human brain is responsible for the cognitive and functional attribute of the human being. The brain is wired with millions of neurons that accept sensory inputs from the outside world which are processed by the brain. Researchers in Machine Learning and Artificial Intelligence in the course of their research figured that the best way to solve problems is by creating a computational system like the human brain. [2] This computational system is called Neural Network - a set of computing models that are designed after biological neural networks - the human brain. Other classical machine learning models have several weaknesses such as solving the real life problems that the neural networks addresses. They can learn and model nonlinear and complicated interactions between inputs and outputs, make generalizations and conclusions, uncover hidden patterns and predictions, and model highly large unstructured data (such as financial time series data). As a result, they are better facilitators of making the best decisions in the areas such as quality control, fraud and financial crime detection, ecological sphere, targeted marketing, logistics and transportation, medical diagnosis, energy forecasting, financial predictions, robotics, and many others. Neural networks algorithms are widely used to process raw pictures and video representations specifically in medical imaging, robotics, facial recognition and image classification. This paper focuses on image classification in order to recognize distinguishable and similar patterns to make accurate predictions using different forms of neural networks. Also, we investigated the mathematical concept of Principal Component Analysis (PCA) that reduce the dimension of large data sets and still retains the majority of the information.

[1]https://www.ncbi.nlm.nih.gov/books/NBK234155/

[2]https://www.sas.com/en_us/insights/analytics/neural-networks.html/

II. General Framework of a Neural Network

A general framework of a neural network architecture is a mapping of an input layer X to a output layer Y. The middle layers or the hidden layers are $x^{(j)}$ with $j-$specifying the sequential ordering. Matrices A_j contains coefficients that map each variable from layer to layer. Dimension of input layer $X \in R^n$ is known. At the entrance of artificial neuron, the inputs are weighted; in the middle section of artificial neuron, a sum function sums all weighted inputs and bias; at the exit of artificial neuron the sum of previously weighted inputs and bias is passing through activation function that is also called transfer function.

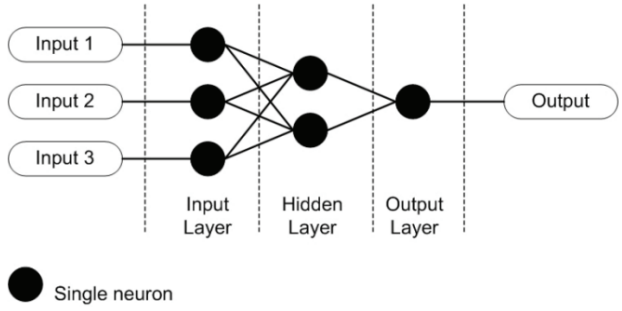

Fig. 1. Representation of a simple artificial neural network.This figure was obtained from Ref [6].

III. Artificial Neural Network

Artificial Neural Networks are commonly confused with neural networks; however, they are not the same thing. In reality artifical neural network, is a form of neural network. In ANN, information passes from one layer to the next without touching a node twice. Thus it is essentially called a feed-forward network. This neural system, which is based on how neurons behave in the brain, recognizes patterns in raw data and assists in the resolution of complex processes. Another parallel with the human brain is that the ANN improves with each new input. The ANN is made up of three or more layers of interconnected nodes, analogous to the brain.

Input, processing, and output data to the deeper layers are the responsibility of all layers. This neural network understands and learns complex things thanks to an inter-layered system like this. Since the theory behind CNN is explained in more detailed and ANN follows the genral framework of a

neural network, we will not cover the theoretical explanation of ANN in much detail.

IV. Theory of Convolutional Neural Network

Convolutional Neural Network otherwise known as CNN or ConvNet, is a branch of Artificial Neural Network that is popular for analyzing visual images. CNN has aided in the scaling of the process by identifying patterns in images using convolutional layer and pooling layer. The CNN typically has three layers - convolutional layer, pooling layer, and the fully-connected layer. Its complexity in comprehending the image grows with each layer. This means that the first layers of the neural network interprets simple elements like edges and colors. The network can distinguish complicated elements such as object forms as the image progresses through layers. The deepest layer can finally recognize the target object. Every layer has a distinct function such as synthesizing, linking, or activating. All the layers in a convolutional neural network is described in more detail below:

- Convolutional layer: Convolutional layer will apply filter to an input image passing the image to the next layer. A convolution combines the values of all the pixels in its receptive area. If you apply a convolution to an image, you will reduce the image size while also combining all of the information in the field into a single pixel. The convolutional layer's final output is a vector. We can employ several types of convolutions depending on the problem we're trying to solve and the features we want to learn. The 2D convolution layer is the most frequent type of convolution, and is generally abbreviated as conv2D. In a conv2D layer, a filter or kernel "slides" through the 2D input data, executing element-wise multiplication. As a result, the findings will be summed into a single output pixel. For each point it slides over, the kernel will do the same procedure, changing a 2D matrix of features into a different 2D matrix of features.

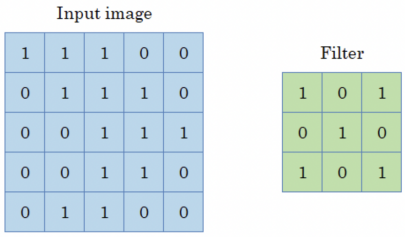

Fig. 2. Representation of input image and a filter to be applied in a convolutional layer. This filter slides across the height and width of the image producing the image representation of that receptive region. This figure was obtained from Ref [3].

- Pooling layer: The concept of pooling is to take groups of pixels, and perform an aggregation over them. Different kinds of pooling are done in a CNN with max pooling, min pooling, and average pooling being the most common ones. In a max pooling, the maximum pixel value of the batch is selected, in a min pooling, the minimum pixel value of the batch is selected where as in an average pooling the average value of all the pixels in the batch is selected.

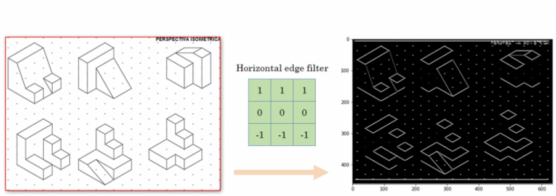

Fig. 3. Representation of a convolutional layer in an image. The convolutional layer extracts only the information we want from the image. In this example, the convolutional layer retains only the horizontal edges. This figure was obtained from Ref [3].

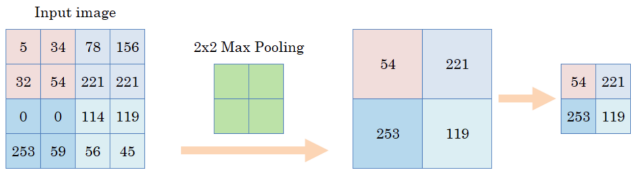

Fig. 4. Representation of a max pooling layer in a CNN. The input image is divided into four $2 * 2$ matrices, and the maximum pixel in each matrix is selected to get a new image. This figure was obtained from Ref [3].

- Fully Connected Layer: Neurons in this layer have full connectivity with all neurons in the preceding and succeeding layer. This layer helps to map the representation between the input and the output.

A combination of convolutional layer, pooling layer, and fully connected layer is used to create a convolutional neural network.

V. Principal Component Analysis

Principal Component Analysis (PCA) is defined as an orthogonal linear transformation that transforms the data to a new coordinate system such that the greatest variance by some scalar projection of the data comes to lie on the first coordinate (called the first principal component), the second greatest variance on the second coordinate, and so on. Mathematically, all we are doing in a PCA is expanding the solution in an orthonormal basis which diagonalizes the underlying system. In simple terms, PCA is a dimensionality-reduction method that reduces the dimensionality of large data sets, by transforming a large set of variables into a smaller one that still contains most of the information in the large set.
The concept of PCA can be performed in 5 sub-steps.

- Standardization: The first step in doing a PCA is to standardize the data. The reason for this is to eliminate variance of initial variables. Variance can skew the contribution that each component provides to analysis. If all of the variables are transformed to same scale, they will contribute equally to the analysis.

- Computation of covariance matrix: We compute a covariance matrix to investigate the relationship between the variables in an input data set. Large covariance means two events have large redundancy while small covariance means little redundancy. The covariance matrix is a pp symmetric matrix (where p is the number of dimensions) that has as entries the covariance associated with all possible pairs of the initial variables.
- Compute the eigenvalue and eigenvector of covariance matrix to compute the principal components: Principal components are the linear combinations of the initial variables in such a way that the principal components are uncorrelated and most of the information within the initial variables is squeezed or compressed into the first components. The first principal component accounts for the largest possible variance in the dataset. The eigenvectors of covariance matrix C_x gives us the directions of the axes where there is the most variance, and eigenvalues gives the amount of variance carried in each principal component.
- Feature vector: Computing the eigenvectors and ordering them by their eigenvalues in descending order, allow us to find the principal components in order of significance. In this step, we choose whether to keep all these components or discard those of lesser significance (of low eigenvalues). So, the feature vector is simply a matrix that has as columns the eigenvectors of the components that we decide to keep.
- Recast the data along the principal component axes: In this step, the aim is to use the feature vector formed using the eigenvectors of the covariance matrix, to reorient the data from the original axes to the ones represented by the principal components.
  This entire section was paraphrased from Ref[9].

VI. Our Dataset

Our dataset is balanced with about 12500 cat and dog images respectively. It was downloaded from Kaggle.Our focus is to perform a binary classification task that can differentiate and classify the pets by observing the characteristics or features each pet possess based on the available images. We see that although both pets have similar features like four legs, two eyes, two ears and so on, there are still noticeable differences between them, for instance, dogs have pointed, flopped or pricked ears while cats have more pronounced whiskers.

VII. Image Classification Using ANN

To perform a binary classification, we used Keras Deep Learning API. Our approach is to structure a dataset of cats and dogs into training and validation folders with class name organized as the folder names. The classifier was trained on about 20K images using Dense layers (layers that are deeply connected with preceding layers). The validation is done with 2.4K images.The images were converted to gray-scale as neural network model expects images to have the same number of input features. For easier analysis, the images were converted to DataFrame using Pandas. Downsizing was introduced so that each image will have the same width and height. Keras has extensive API and library to compile and train a model as well as to store the model. We used the $model.compile$ and $model.fit$ to achieve this. Loss function was set to $categorical\_crossentropy$ because this loss function works the best for multi-class classification model where two or more output labels is required. The output label is assigned one-hot category encoding value in form of 0s and 1. The output activation function was set to $softmax$ function. $Softmax$ converts a vector of values to a probability distribution. The elements of the output vector are in range $(0, 1)$ and sum to 1. Finally, Epoch was set to 50 for the initial run then 100 and 500 to improve accuracy and reduce loss. This comes with a performance trade off as the processing time from the neural network traversing increases (CPU processing approach is slower).

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_20 (Dense)            (None, 2048)              18876416

 dense_21 (Dense)            (None, 1024)              2098176

 dense_22 (Dense)            (None, 512)               524800

 dense_23 (Dense)            (None, 128)               65664

 dense_24 (Dense)            (None, 2)                 258

=================================================================
Total params: 21,565,314
Trainable params: 21,565,314
Non-trainable params: 0
_________________________________________________________________
```

Fig. 5. Artificial neural network model for binary classification of cats and dogs.

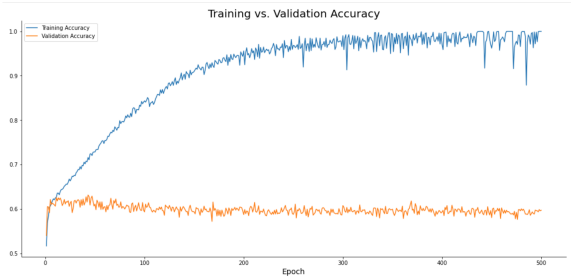

Fig. 6. Graph that shows the training vs validation accuracy for our ANN. We can see that the validation accuracy plateaus around 60%.

VIII. Image Classification Using CNN

To improve the performance on our ANN, we created a CNN. The construction of CNN model follows the same pattern as of ANN. Like in ANN, we perform a binary classification using Keras Deep Learning API, but with the addition of two important layers: convolutional layer and pooling layer. First of all, we processed the image using built-in image processor function in python's Tensorflow package. Like in ANN, Keras has extensive API and library to compile and train a model as well as to store the model. We used the

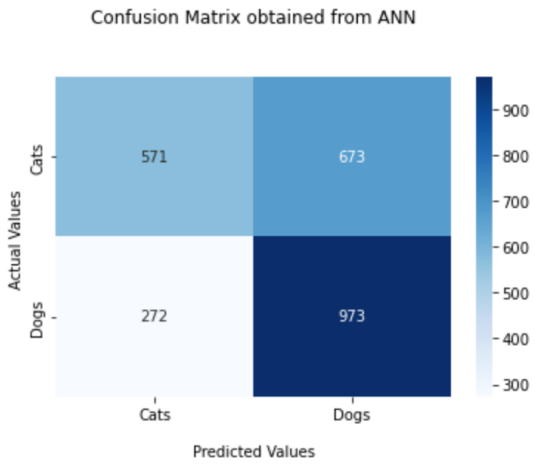

Fig. 7. Confusion matrix we obtained from our ANN model when predicting images of cats and dogs. Our model correctly predicted cats only 571 times out of 1244 pictures of cats. Similarly, our model correctly predicted dogs 973 times out of 1245 pictures of dogs. We got only 45% accuracy on cats whereas our accuracy was 78% on dogs.

$model.compile$ and $model.fit$ to achieve this. Like in ANN, loss function was set to $categorical\_crossentropy$, output activation function was set to $softmax$ function. Finally, Epoch was set to 10. Due to extremely high computational necessity of running the CNN model, we did not try it for 50, 100, and 500 epochs.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 222, 222, 32)      896

 max_pooling2d (MaxPooling2D  (None, 111, 111, 32)     0
 )

 conv2d_1 (Conv2D)           (None, 109, 109, 32)      9248

 max_pooling2d_1 (MaxPooling  (None, 55, 55, 32)       0
 2D)

 flatten (Flatten)           (None, 96800)             0

 dense (Dense)               (None, 128)               12390528

 dense_1 (Dense)             (None, 2)                 258

=================================================================
Total params: 12,400,930
Trainable params: 12,400,930
Non-trainable params: 0
```

Fig. 8. Convolutional neural network model for binary classification of cats and dogs.

IX. DIMENSIONAL REDUCTION USING PCA AND IMAGE CLASSIFICATION

We used the concept of PCA to develop a machine learning algorithm that can recognize picture of a dog or a cat from the data set. To develop the algorithm, first of all we loaded the image data set of cats and dogs and converted them into *pandas* dataframe in python. We then have to separate our training set and testing set for which we used the blackbox function in python: *scikit.learn*. This function splits the input

Fig. 9. Graph that shows the training vs validation loss for our CNN on running for 10 epochs.

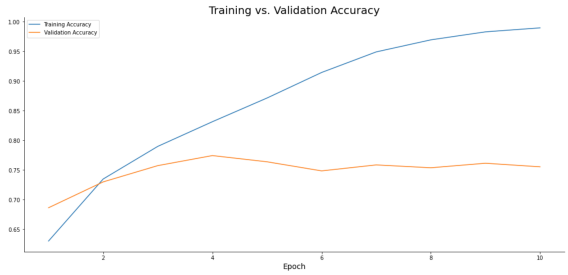

Fig. 10. Graph that shows the training vs validation accuracy for our CNN on runing for 10 epochs. We can see that the validation accuracy is around 70% which is a huge improvement on our ANN model.

```
Epoch 1/10
313/313 [==============================] - 629s 2s/step - loss: 0.6865 - accuracy: 0.6304 - val_loss: 0.5962 - val_ac
curacy: 0.6864
Epoch 2/10
313/313 [==============================] - 600s 2s/step - loss: 0.5242 - accuracy: 0.7349 - val_loss: 0.5199 - val_ac
curacy: 0.7300
Epoch 3/10
313/313 [==============================] - 636s 2s/step - loss: 0.4458 - accuracy: 0.7899 - val_loss: 0.5036 - val_ac
curacy: 0.7575
Epoch 4/10
313/313 [==============================] - 534s 2s/step - loss: 0.3735 - accuracy: 0.8315 - val_loss: 0.5014 - val_ac
curacy: 0.7744
Epoch 5/10
313/313 [==============================] - 558s 2s/step - loss: 0.2981 - accuracy: 0.8714 - val_loss: 0.5242 - val_ac
curacy: 0.7639
Epoch 6/10
313/313 [==============================] - 561s 2s/step - loss: 0.2117 - accuracy: 0.9145 - val_loss: 0.6145 - val_ac
curacy: 0.7486
Epoch 7/10
313/313 [==============================] - 562s 2s/step - loss: 0.1363 - accuracy: 0.9492 - val_loss: 0.7414 - val_ac
curacy: 0.7587
Epoch 8/10
313/313 [==============================] - 607s 2s/step - loss: 0.0850 - accuracy: 0.9695 - val_loss: 0.8894 - val_ac
curacy: 0.7538
Epoch 9/10
313/313 [==============================] - 664s 2s/step - loss: 0.0519 - accuracy: 0.9830 - val_loss: 1.1221 - val_ac
curacy: 0.7615
Epoch 10/10
313/313 [==============================] - 700s 2s/step - loss: 0.0367 - accuracy: 0.9898 - val_loss: 1.2062 - val_ac
curacy: 0.7554
```

Fig. 11. This figure shows the running of our CNN model on 10 epochs. We can see the validation accuracy is around 75%.

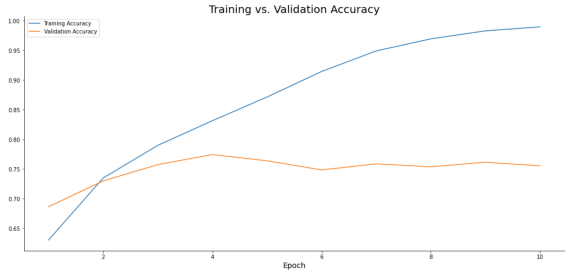

Fig. 12. Graph that shows the training vs validation accuracy for our CNN on running for 10 epochs. We can see that the validation accuracy is around 70% which is a huge improvement on our ANN model.

space into training data and test data; features and labels. We then performed PCA and used 200 components from the *scikit.learn*. The next step is to transform the training data into the PCA. Then we create a classifier class of Support

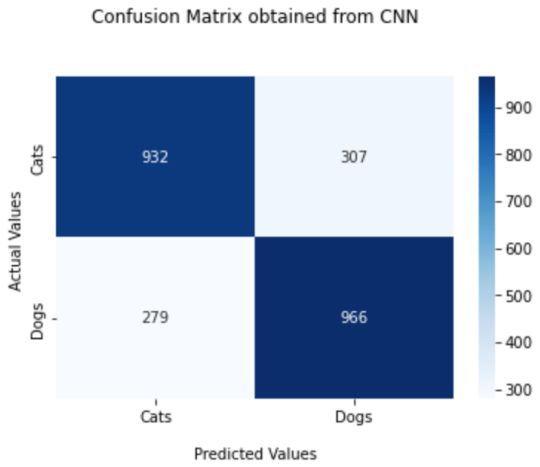

Fig. 13. Confusion matrix we obtained from our CNN model when predicting images of cats and dogs. Our model correctly predicted cats 932 times out of 1239 pictures of cats. Similarly, our model correctly predicted dogs 966 times out of 1245 pictures of dogs. We got 75% accuracy on cats whereas our accuracy was 77.5% on dogs. This is a significant improvement from our CNN model.

Vector Classifier (SVC) and fit the created training data on the classifier. A bit of trial and error is needed at this point to create the best classifier fit. We experimented with linear kernel, default non-linear kernel given by 'rgb, sigmoid kernel, and pre-computed kernel. After experimentation, the regularization parameter was set to 1000, and the kernel coefficient was set to 0.001. Now we pass test dataset to the classifier, and perform our prediction. Now we print our prediction report.

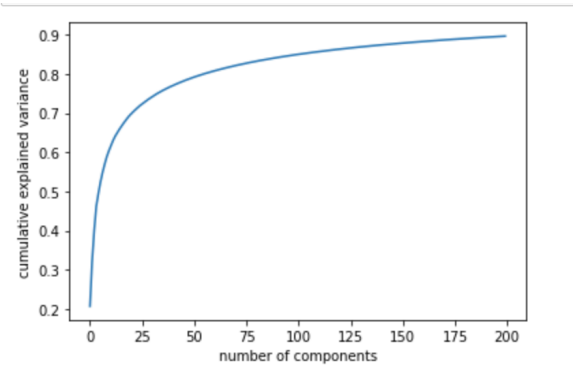

Fig. 14. The variance ratio parameter returns a vector of the variance explained by each dimension. This graph plots the cumulative variance explained solely by the $i + 1$st dimension.

X. Conclusion

ANN is a feed forward Algorithm that has a significant amount of drawbacks especially with requiring large amounts of data and heavily relying on trial and error technique. CNN on the other hand introduces several layers to the training module such as pooling and convolution that improves prediction by requiring less human supervision. Both methods requires a large amount of resource and time to train datasets. However

```
Predicting cats or dogs on the test set
done in 15.371s
              precision    recall  f1-score   support

         cat       0.63      0.61      0.62      2493
         dog       0.62      0.64      0.63      2515

    accuracy                           0.63      5008
   macro avg       0.63      0.63      0.63      5008
weighted avg       0.63      0.63      0.63      5008
```

Fig. 15. This figure shows the prediction of all cats and dogs in the dataset. Precision column shows the accuracy in our prediction, whereas recall shows the number of times our model was correct in our prediction. There is a trade-off between precision and recall, and f-score is an indicator how well the model worked on both precision and recall. We obtained 63% accuracy using just 200 PCA components which is comparable to our ANN model.

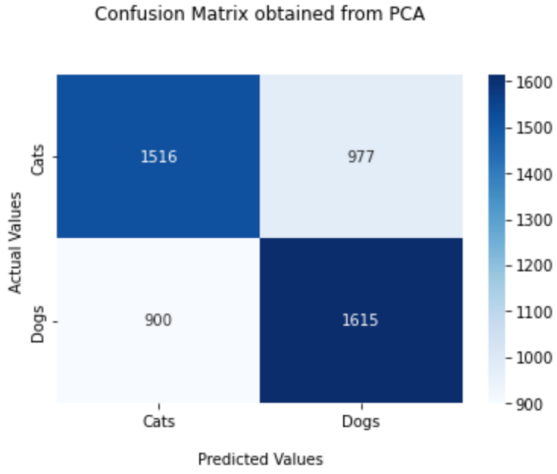

Fig. 16. Confusion matrix we obtained from our PCA model when predicting images of cats and dogs. Our model correctly predicted cats 1516 times out of 2493 pictures of cats. Similarly, our model correctly predicted dogs 1615 times out of 2515 pictures of dogs. We got 63% precision on cats whereas our precision was 62% on dogs. This is as good as our ANN model.

the introduction of PCA subsequently improves the time it takes to train these by reducing the image data substantially without having to distort information contained in the images.

XI. References

[1] Memon, M. (2022, April 11). Neural networks: CNN vs ANN VS RNN. Levity. Retrieved May 11, 2022, from https://levity.ai/blog/neural-networks-cnn-ann-rnn

[2] What is a convolutional layer? Databricks. (2020, May 15). Retrieved May 11, 2022, from https://databricks.com/glossary/convolutional-layer

[3] Zafra, M. F. (2020, May 25). Understanding convolutions and pooling in Neural Networks: A simple explanation. Medium. Retrieved May 11, 2022, from https://towardsdatascience.com/understanding-convolutions-and-pooling-in-neural-networks-a-simple-explanation-885a2d78f211

[4] Brownlee, J. (2019, July 5). A gentle introduction to pooling layers for Convolutional Neural Networks. Machine Learning Mastery. Retrieved May 11, 2022, from https://machinelearningmastery.com/pooling-layers-for-convolutional-neural-networks/

[5] Saha Sumit (Dec 15, 2018). A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way. Medium. Retrieved May 11, 2022, from https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

[6] Krenker, A and Bester, J and Kos, A. (2011) Introduction to the Artificial Neural Networks. Krenker2011IntroductionTT

[7] Ackerman S. Discovering the Brain. Washington (DC): National Academies Press (US); 1992. Foreword. Available from: https://www.ncbi.nlm.nih.gov/books/NBK234155/

[8] Zafra Miguel (May 25, 2020). Understanding Convolutions and Pooling in Neural Networks: a simple explanation. Medium. https://towardsdatascience.com/understanding-convolutions-and-pooling-in-neural-networks-a-simple-explanation-885a2d78f211

[9] Jaadi, Z. (2021, December 1). A step-by-step explanation of principal component analysis (PCA). Built In. Retrieved May 12, 2022, from https://builtin.com/data-science/step-step-explanation-principal-component-analysis