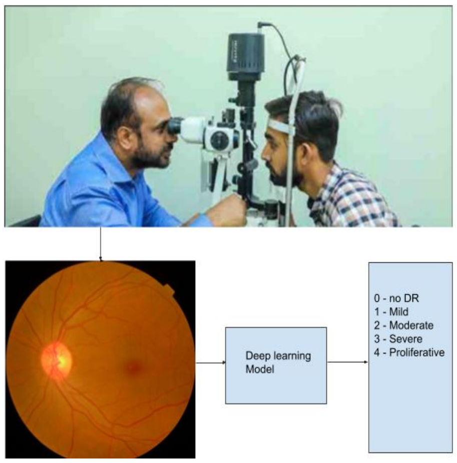
# Neural Networks for Diabetic Retinopathy detection

Gouri Kulkarni CSML1020 - Machine Learning at Scale, York University School of Continuing Education

***Abstract*—After taking the course [3], this article is my attempt to implement convolutional neural networks to detect Diabetic Retinopathy in fundus images obtained from a Kaggle dataset.[9].**

**Background: Citing from the IDF Diabetes Atlas 9th Edition, there are 463 million people living with diabetes today. By 2045, 700 million people will have the disease worldwide [6]. Diabetic Retinopathy is an eye condition that can cause vision loss in people who have diabetes. It affects blood vessels in the retina , the light-sensitive layer of tissue in the back of the eye. Diabetes patients must get a comprehensive dilated eye examination at least once a year. The most common form of screening is manual examination of images of the retina captured using fundus cameras. Symptoms may not show at first, but early detection can help delay vision loss. In later stages,blood vessels start to bleed into the vitreous gel like fluid in the center of the eye. Other problems can also be prevented or treated if diabetic retinopathy is detected early. The size of the data poses several challenges on all involved stakeholders. Machine learning has been used for years to speed up health informatics.Intelligent algorithms have been developed that perform important visual perception tasks such as object recognition and categorization [2]. Convolutional Neural Networks allow for classification of images, given a sufficiently large dataset. The dataset provided by kaggle competition APTOS 2019 provides us with 5590 labelled fundus images.A simple CNN network and pretrained VGG16 networks were implemented.The model was trained on google colab for deployment on GCP.**

***Index Terms*—ComputerVision, DeepLearning, Diabetic Retinopathy, CNN, Machine Learning, Big Data**



## I. Introduction

Regular screening is key to preventing diabetic retinopathy. Timely diagnosis can help reduce the number of patients who suffer vision loss before diagnosis. Poor turnaround time can negatively impact patient care and can have life changing consequences for patients A misdiagnosis can delay sending a patient to an eye test by a year. The link outlines steps to define the problem.[5] Machine learning solutions that learn from experience with respect to the task of classifying fundus images and evaluated against standards and performance measures can mitigate these issues by allowing opthalmologists to focus their time on high risk cases, providing more information to correct potential misdiagnoses and helping with effective case management. The dataset used for this project has been sourced from Kaggle [9]. Section 6 explains the Neural network architectures and the basis for selecting CNN. The input to our algorithms are labelled .png images.This is a multi classification problem. Some of the challenges posed by the large number of diabetic patients are :

1) Determining how often a patient needs an eye exam depends on the patient's risk of eye disease
2) Time - to treatment periods
3) Self-management for people with diabetes
4) Cost to the health system
5) Variance in data
6) Labelling the data
7) Complexity of data
8) Adversarial attacks

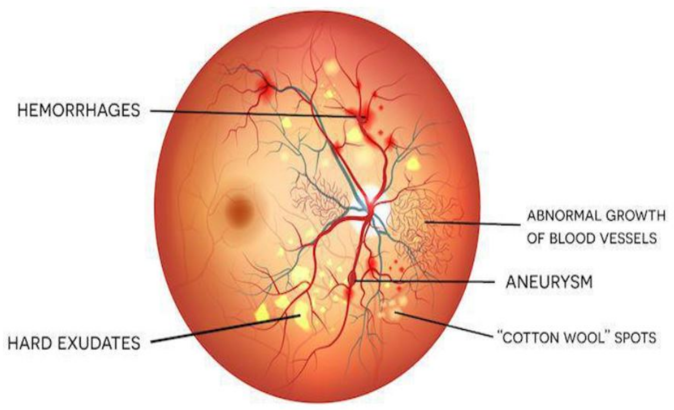
The labels used by opthalmologists are

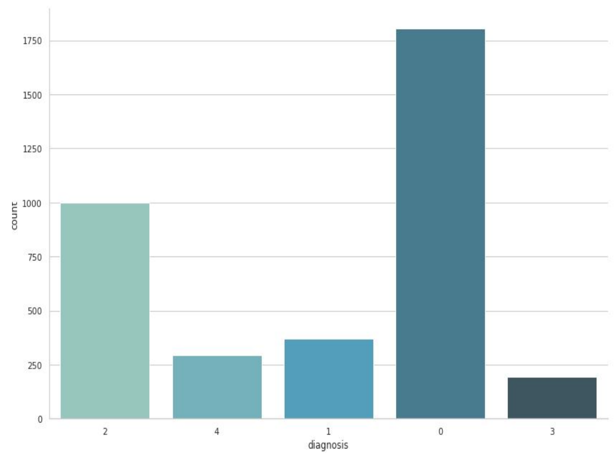
- 0 No DR
- 1 Mild
- 2 Moderate
- 3 Severe
- 4 Proliferative DR

The solution can be extended to solve other problems

1) Diabetic Macular edema - when blood vessels in the retina leak fluid , causing swelling in the macula
2) Neovascular glaucoma - when abnormal blood vessels grow out of the retina and block fluid from draining out of the eye
3) Retinal detachment - when scars from in the back of the eye and pull the retina from the back of the eye

The solution to this problem can be used to develop commercial devices for diabetic retinopathy detection.





## II. Related Work

This project [9] was based on previous work done by kaggle contributors. The first Kaggle competition was held in 2015. The computational resources available today make it possible to use the algorithms developed in the past 35 years on complex scientific images. Google Research has been conducting projects on Diabetic Retinopathy and in 2016 published results in [1] which showed that the prediction scores of deep learning algorithms surpassed the score of general opthalmologists in detecting diabetic retinopathy, eliminating human error.In 2017 , Google developed a model on par with retinal specialists by training it on a consensus of labels generated by many opthalmologists. The strategies used by top scorers of the Kaggle competition [9] include combining the 2015 and 2019 datasets, ensembling, picking top performimg epochs from the training process.

## III. Dataset and data aggregation

The data was acquired from Kaggle and downloaded to a google drive to be used in colab notebooks. GPU runtimes were selected.

The labels are

- 0 No DR
- 1 Mild
- 2 Moderate
- 3 Severe
- 4 Proliferative DR

## IV. Preprocessing

The images vary in size. I did not resize the images for the first iteration. The images were scaled. A simple baseline CNN was trained and a VGG16 pre-trained model was fine tuned. Data augmentation methods will be explored in a future iteration [4] Data preparation requirements of pretrained models vary. 1.Read the picture files 2.Decode PNG content to RGB pixels 3.Convert this into floating tensors 4.Rescale pixel values (between 0 to 255) to [0,1] interval. I made use of the ImageDataGenerator method available in keras to do all the preprocessing

As the images are clustered in one folder and images belonging to all classes reside in one folder, and we have a text file that contains the label information, I have created a dataframe using pandas and text files provided, with the image file name and labels as columns.

## V. Data Augmentation

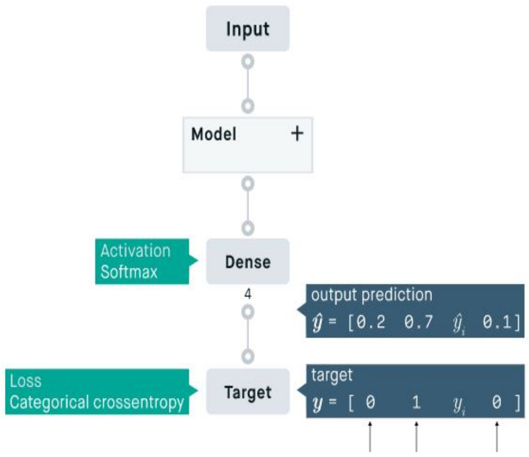
Data augmentation techniques can be implemented to improve model performance , reduce generalization error and improve predictive performance on test data. Data augmentation requirements of pretrained models vary.

## VI. Network Architectures

Neural networks make a prediction even when given the wrong data [14].The challenge is to use the right neural network to solve the problem. Three classes of artificial neural networks are: Multilayer Perceptrons (MLPs), Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs). MLPs are suitable for tabular datasets , classification and regression problems. They can learn mappings from inputs to outputs. Applying MLPs to image data requires the pixels of each image be reduced down to one long row of data. This is a tedious process and was not pursued. CNNs were designed to map image data to an output variable and are the go-to method for a prediction problem involving image data. CNNs develop an internal representation of a two dimensional image. This allows the model to learn position and scale in variant structures in the data. Spatial relationships can also be captured. RNNs work best with sequence prediction problems. RNNs are not appropriate for image data. A simple CNN model was trained, and VGG16 pre-trained models were fine tuned using Tensorflow and Keras on Google Colab GPU. Though it was time consuming, the models were trained on the entire dataset in order to catch as many features as possible and understand some of the challenges posed by Big Data. Due to time constraints, few epochs were run, but better observations could have been made with more epochs.

## VII. Loss function

As this is a multi-class classification task, the loss function used is categorical crossentropy. A fundus image can only belong to one out of the possible categories. The model will decide which one. It quantifies the difference between two probability distributions, the output prediction and the target.



The categorical crossentropy loss function calculates the loss of an example by computing the following sum:

$$- \sum_{i=1}^{outputsize} y_i.log\hat{y_i}$$

$\hat{y_i}$ is the $i^{th}$ scalar value in the model output , $y_i$ is the corresponding target value and output size is the number of scalar values in the model output. The loss measures how distinguishable two discrete probability distributions are from each other.

The loss gets smaller as the distributions get closer to each other. $y_i$ is the probability that event $i$ occurs and the sum of all $y_i$ is 1 , meaning that exactly one event may occur. The models used will give a high probability to the correct label and a low probability to the other labels. Softmax is the activation function used with categorical cross entropy loss function.

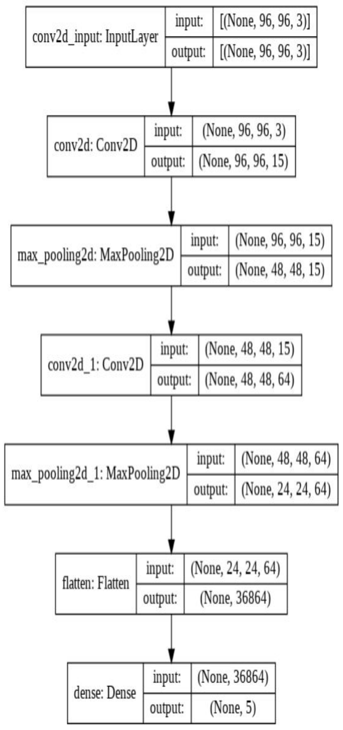
## VIII. Image augmentation

Though we have 5000+ images, the dataset is still small and does not give the models enough to train on. With image augmentation, the size if the training dataset can be artificially expanded by creating modified versions of the dataset. What transforms should be applied depends on the training images. Possible transformations are horizontal flip, vertical flip, zoom, shift, flip. Having domain knowledge can help with making the decision on appropriate image augmentation strategies. Whereas data preparation like image resizing is applied to the entire dataset, image augmentation is only applied to the training dataset. Strategies for image augmentation in object detection have been discussed in a paper [7]. An optimized image augmentation strategy can improve accuracy.

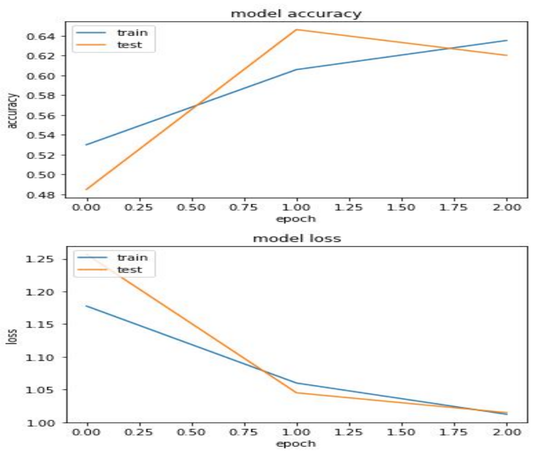
## IX. Models

### A. Baseline CNN

The image size used is 96 width, 96 height and 3 channels.Images were not resized before feeding them to the model. The layers implemented are two convolutional layers with 15 and 64 output filters and kernel size 3 by 3, two max pooling layers and one flattening layer. The final dense layer with 5 neurons with softmax activation gives a probability distribution for each output. Total number of parameters in the model is 193,449 The metric used is accuracy and learning rate used is .01 . Used Early stopping to stop training when a monitored metric ( accuracy) has stopped improving and Mode Checkpoint we save the best weights. Architecture [ baseline-architecture] Training was run for 3 epochs, though running more epochs would have given better observations. Time taken to train 3 epochs was over an hour. The model was saved after every epoch. The highest accuracy on the validation set was reached in the second epoch.


conv2d_input: InputLayer
input: [(None, 96, 96, 3)]
output: [(None, 96, 96, 3)]
conv2d: Conv2D
input: (None, 96, 96, 3)
output: (None, 96, 96, 15)
max_pooling2d: MaxPooling2D
input: (None, 96, 96, 15)
output: (None, 48, 48, 15)
conv2d_1: Conv2D
input: (None, 48, 48, 15)
output: (None, 48, 48, 64)
max_pooling2d_1: MaxPooling2D
input: (None, 48, 48, 64)
output: (None, 24, 24, 64)
flatten: Flatten
input: (None, 24, 24, 64)
output: (None, 36864)
dense: Dense
input: (None, 36864)
output: (None, 5)

Learning curves generated are reproduced below.



### B. Pretained models

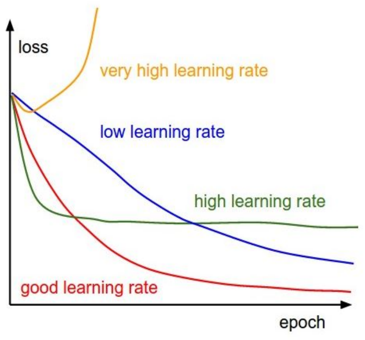
Training can take long, depending on the size and complexity of the images. Re-using weights from pre-trained models developed for the ImageNet dataset , we download the model model and use it 1. as is 2. freezing some layers. [12] Transfer learning involves using models trained on one problem as a starting point on a related problem. Transfer learning is flexible, allowing the use of pre-trained models directly, as feature extraction preprocessing, and integrated into entirely new models. Keras provides convenient access to many top performing models on the ImageNet image recognition tasks such as VGG, Inception, and ResNet. Layers closer to the input layer learn low level features such as lines , layers in the middle learn complex abstract features and layers closer to the output interpret extracted features in the context of a classification task. The task of classifying fundus images is quite different from classifying general objects.The output from the initial layers would be more appropriate. Various usage patterns are possible.

- use directly
- use a portion to pre process images and extract relevant features
- freeze layers of pretrained model and integrate into new model
- integrate pretrained model into new model

Transfer learning decreases the training time and decreases generalization error. The pretrained models are available via the Application API and includes functions to load and prepare the data. Global pooling layers can be added to specify pooling output. Images can be prepared using the pre processing functions so that pixels are scaled just as the input images to the pretrained model. Learning rate: by how much we are adjusting the weights of our network with respect to the loss gradient , step size [10].

new weight = existing weight - learning rate * gradient

Learning rates are configured by intuition or based on past experience The learning rate controls how quickly the model is adapted to the problem. Smaller learning rates require more training epochs given the smaller changes made to the weights each update, whereas larger learning rates result in rapid changes and require fewer training epoch.A learning rate that is too large can cause the model to converge too quickly to a suboptimal solution, whereas a learning rate that is too small can cause the process to get stuck.I have tried to experiment with learning rates.

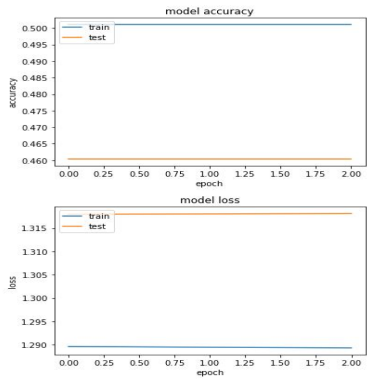


Effect of various learning rates on convergence (Img Credit: cs231n)

Lower training times mean lesser training costs on GPU cloud compute. Plotting learning rate vs loss over iterations, the optimal learning rate can be located.

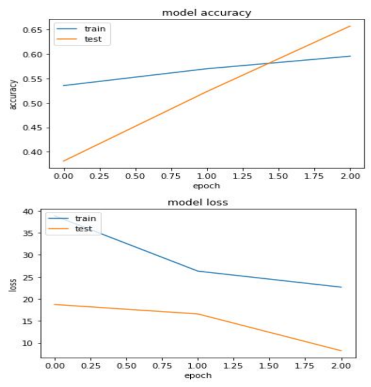
### C. VGG16 Pre trained model

The paper [13] Images used for models pre trained on VGG16 have to be rescaled to 224X224. The target image size was set to 224. The weights and parameters of the VGG16 model were frozen so that they were not trained again.All layers were added to the model, except the all layers from the pretrained model were set to be non trainable. The last layer has 5 output classes and uses softmax activation. The model was complied with a learning rate of .01 and fitted for 3 epochs. Time taken to train 3 epochs was over an hour. With a learning rate .01 over 3 epochs we do not see any convergence

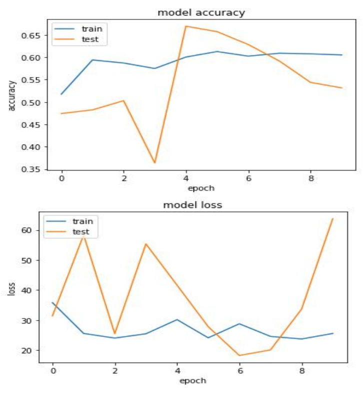


Next , the model was compiled with a learning rate of .1 and fitted for 3 epochs. The time taken to train the model was about the same.

From these graphs, it is clear that more epochs need to be observed The model loss is also very high. The model with .1 learning rate took thrice the time to train. over the model with .01 learning rate.

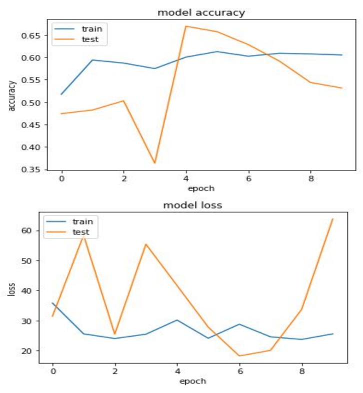


The results from running 10 epochs and a learning rate .1 are as follows.

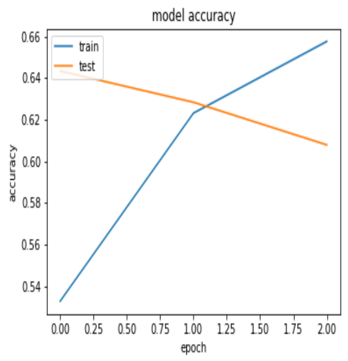


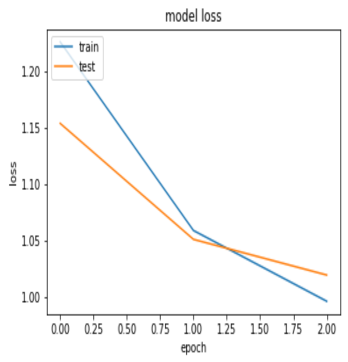
With a learning rate of .1 the model loss was very high and the accuracy was comparable to learning rate .01 . We can move on to iterations involving image augmentation.

The results from running 5 epochs and a learning rate of ,01 are as follows:

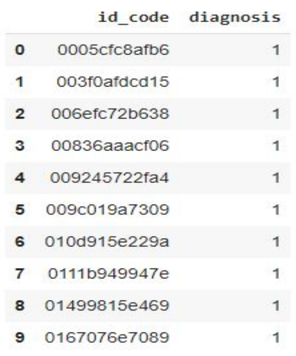


## X. VGG16 Model deployment





Complied with a learning rate .0001



| | id_code | diagnosis |
|---|---|---|
| 0 | 0005cfc8afb6 | 1 |
| 1 | 003f0afdcd15 | 1 |
| 2 | 006efc72b638 | 1 |
| 3 | 00836aaacf06 | 1 |
| 4 | 009245722fa4 | 1 |
| 5 | 009c019a7309 | 1 |
| 6 | 010d915e229a | 1 |
| 7 | 0111b949947e | 1 |
| 8 | 01499815e469 | 1 |
| 9 | 0167076e7089 | 1 |

Predictions from the deployed model - Version 1

Version 1 gave a maximum validation accuracy of 64 percent with a loss of 1.2 percent. The model predicts all test images as 1 - Mild. Further experiments and versions, implementing appropriate pre processing for VGG16, image resizing, data augmentation can be conducted and models can deployed on AI Platform. The model version 1, stored on a google storage bucket has a size 537478729 bytes, which exceeds the allowed maximum size of 524288000 bytes and could not be saved on a single core CPU. The SavedModel must be 500 MB or smaller in order to deploy deploy it to a model version that uses a legacy (MLS1) machine type. It can be up to 2 GB if we use a Compute Engine (N1) machine type. Machine types for online prediction AI Platform

Prediction allocates nodes to handle online prediction requests sent to a model version. When we deploy a model version, we can customize the type of virtual machine that AI Platform Prediction uses for these nodes.

Machine types differ in several ways:

- Number of virtual CPUs (vCPUs) per node
- Amount of memory per node
- Support for GPUs, which can be added to some machine types
- Support for certain AI Platform Prediction features
- Pricing
- Service level agreement (SLA) coverage

By selecting a machine type with more computing resources, we can serve predictions with lower latency or handle more prediction requests at the same time. A summary of available machine types for AI Prediction is here [11]

## XI. Results

| Model | Accuracy |
|---|---|
| Simple CNN with lr .01 and 3 epochs | 53.00 |
| Pretrained VGG16 with lr .01 and 3 epochs | 46.00 |
| Pretrained VGG16 with lr .1 and 3 epochs | 65.00 |
| Pretrained VGG16 with lr .1 and 10 epochs | 66.9 |
| Pretrained VGG16 with lr .0001 and 3 epochs | 64.34 |

## XII. Conclusion

In this project, we started by ingesting the data from Kaggle to google drive for use with google colab. A simple CNN and several VGG16 models were fine tuned with different values of learning rate and with/without freezing the pretrained layers. Tensorflow/Keras runs on a single GPU on Google Colab with no code configuration required, and was used for experimentation. To use TPUs to run image classification, the image format needs to be converted to TFRecords [8]. The models (on GPU) did not achieve high accuracy. The reasons behind this could be many and are the subject of future work.

## XIII. Code

The code developed has been posted on a GitHub repository. https://github.com/csml1020gk/project

## XIV. Acknowledgments

I am grateful to Prof. Karthik Kuber for his mentorship, guidance, encouragement, sharing of best practices and valuable insights into deep learning and big data challenges over the past two months. One of the takeaways from this course is the development of key skills necessary to be successful in future data science endeavours.

## References

[1] URL: https://jamanetwork.com/journals/jamaophthalmology/article-abstract/2771167?resultClick=1.

[2] *Computer Vision*. URL: https://www.sciencedirect.com/science/article/pii/S1474667015378903?via%3Dihub.

[3] *CSML1020*. URL: https://learn.continue.yorku.ca/course/view.php?id=4613.

[4] *Data Augmentation*. URL: https://machinelearningmastery.com/best-practices-for-preparing-and-augmenting-image-data-for-convolutional-neural-networks/.

[5] *Define the problem*. URL: https://machinelearningmastery.com/how-to-define-your-machine-learning-problem/.

[6] *IDF Diabetes Atlas 9th Edition*. URL: https://www.diabetesatlas.org/en/.

[7] *Image Augmentation*. URL: https://arxiv.org/abs/1906.11172.

[8] *Implementing on TPUs*. URL: https://keras.io/examples/vision/xray_classification_with_tpus/.

[9] *Kaggle Competition*. URL: https://www.kaggle.com/c/aptos2019-blindness-detection.

[10] *Learning Rate*. URL: https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/.

[11] *Machine types*. URL: https://cloud.google.com/ai-platform/prediction/docs/machine-types-online-prediction.

[12] *Transfer Learning*. URL: https://machinelearningmastery.com/how-to-use-transfer-learning-when-developing-convolutional-neural-network-models/.

[13] *VGG16*. URL: https://arxiv.org/abs/1409.1556.

[14] *When to use MLP, CNN, RNN networks*. URL: https://machinelearningmastery.com/when-to-use-mlp-cnn-and-rnn-neural-networks/.