UNIVERSITY PARTNER
UNIVERSITY OF WOLVERHAMPTON
iAcademy
Quality Education Improving Lives

HERALD COLLEGE KATHMANDU

Academic Year	Module	Portfolio Project	Assessment Type
2022	6CS012 - Artificial Intelligence and Machine Learning	2	Individual Report

Emotion Detection
(Image Classification using Deep Learning)

Student ID : NP03A190031
University ID : 2050058
Student Name : Ankit Tamrakar
Section : L6CG8
Submitted on : 02-05-2022

Abstract

The dataset we used is made up of 35,685 grayscale pictures of faces with a resolution of 48x48 pixels, divided into two groups: train and test. To categorize the photos, the emotion portrayed by the faces are used. Image classification refers to task of classifying groups of digital pixels or vectors within an image based on a set of criteria. We trained a model using one input layer, one DenseNet layer, one Pooling layer, three Dense layers, and a classification layer at the end. During the development of the model, we used a DenseNet layer. The model's accuracy cannot be expected to be particularly good because the data set contains photos of poor quality and small size. Because the dataset contains relatively small photographs and an unequal distribution of classes, it's tough to come up with more reliable results.

Contents

1. Introduction

The dataset we've selected focuses on emotion detection. Emotions are mental states induced by neurophysiological changes. They are related with a variety of ideas, sensations, behavioral responses, and a degree of pleasure or unhappiness.

We used a image dataset which consists of 35,685 grayscale photographs of faces with a resolution of 48*48 pixels. The dataset comes divided originally into a train and test dataset. The emotion conveyed by the human faces and expressions are utilized to classify the images into 7 major human emotions (happiness, neutral, sadness, anger, surprise, disgust, fear).

Image classification is the task of classifying cluster of digital coordinates i.e., pixels / vectors which exists in a digital image using a definite set of rules. The categorization law can be applied to the image using one or more than one spectral or textural characterizations. (Boesch, 2022)

1.1 Aims

i) To develop an image classification system based on CNN
ii) Use google collab to collaborate and develop a machine learning model

1.2 Objectives

i) Perform some basic manipulation of digital images
ii) Design, build, and train CNN model for image classification process
iii) Familiarize myself with libraries like TensorFlow Keras.

1.3 CNN

Convolutional Neural Networks (CNNs) are a type of Deep Learning Network that accepts an image as input, assign importance to various landmarks/features within the image, and discriminate them. It utilizes learnable weights and biases to successfully replicate a human neural system. A CNN requires comparatively very less amount of pre-processing than traditional methods of classification. While simple approaches require manual selection of filters, different models have been developed to help CNNs learn the weights and biases by itself with adequate training. A CNN's architecture was inspired by connectivity pattern of neurons in the human brain. (Saha, 2018)

CNN is widely utilized for image classification. It is because it examines every landmark of the image individually. As a result, when it comes to recognizing similar features, CNN outperforms all the other existing image comparison algorithms. It also requires large amount of training time and large dataset, resulting in less amount of errors and very high accuracy.

2. Methodology

2.1 Model summary:

```
model.summary()

Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 48, 48, 3)]       0

 densenet169 (Functional)    (None, 1, 1, 1664)        12642880

 global_average_pooling2d (G  (None, 1664)             0
 lobalAveragePooling2D)

 dense (Dense)               (None, 256)               426240

 dropout (Dropout)           (None, 256)               0

 dense_1 (Dense)             (None, 1024)              263168

 dropout_1 (Dropout)         (None, 1024)              0

 dense_2 (Dense)             (None, 512)               524800

 dropout_2 (Dropout)         (None, 512)               0

 classification (Dense)      (None, 7)                 3591

=================================================================
Total params: 13,860,679
Trainable params: 1,217,799
Non-trainable params: 12,642,880
_________________________________________________________________
```

Figure 1: Model Summary

The model we have trained has 1 input layer, 1 DenseNet Layer, 1 Pooling Layer and 3 Dense Layers along with a classification layer at the end.
We have implemented a DenseNet layer while developing the model. A DenseNet is a type of CNN that uses Dense Blocks to create strong connections between each layers, with all levels directly associated with each other. To keep and preserve the algorithm's feed-forward nature, each layer are provided with extra inputs from all preceding levels and sends its own feature-maps to all subsequent layers. (Huang, et al., 2017)
Hyperparameters provided to the model are:

```
HYPERPARAMETRERS AND DIRECTORIES

SEED = 12
IMG_HEIGHT = 48
IMG_WIDTH = 48
BATCH_SIZE = 64
EPOCHS = 30
FINE_TUNING_EPOCHS = 20
LR = 0.01
NUM_CLASSES = 7
EARLY_STOPPING_CRITERIA=3
CLASS_LABELS  = ['Anger', 'Disgust', 'Fear', 'Happy', 'Neutral', 'Sadness', "Surprise"]
CLASS_LABELS_EMOJIS = ["😡", "🤢" , "😱" , "😊" , "😐 ", "😔" , "😲" ]
```

Figure 2: Hyperparameters

There are total of 1217799 number of trainable hyperparameters, and 12642880 non trainable parameters.

2.2 Training of the model:

2.2.1 Loss Functions:

The loss function in a neural network measures the difference between the predicted output and the outcome attained by implementing different machine learning methods. The average of all losses is used to compute this cost. Any gradient that are used to adjust the weight of the neural network may be obtained using the loss function.

```
inputs = tf.keras.layers.Input(shape=(IMG_HEIGHT ,IMG_WIDTH,3))
classification_output = final_model(inputs)
model = tf.keras.Model(inputs=inputs, outputs = classification_output)

model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001),
              loss='categorical_crossentropy',
              metrics = ['accuracy'])

return model
```

Figure 3: Loss Function

We used categorical cross-entropy as loss function for the model. As a loss function, cross-entropy is very frequently used as a metric based on entropy in the field of machine learning. It relies on entropy and calculates the difference between distribution of probabilties. (Brownlee, 2020)

2.2.2 Optimization Algorithm:

```
inputs = tf.keras.layers.Input(shape=(IMG_HEIGHT ,IMG_WIDTH,3))
classification_output = final_model(inputs)
model = tf.keras.Model(inputs=inputs, outputs = classification_output)

model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001),
              loss='categorical_crossentropy',
              metrics = ['accuracy'])

return model
```

Figure 4: Adam Optimization

Adam optimizer was used for compiling the model. The Adam optimization method is variation of SGD classifier that has recently gained momentum in areas such as computer vision and natural language processing. The algorithm is considered among the best till date and widely used in case of large number of parameters and/or input data. It self-optimizes itself and requires very less efficiency and computation.

Fine Tuning

```
# Un-Freezing the feature extraction layers for fine tuning
model.layers[1].trainable = True

model.compile(optimizer=tf.keras.optimizers.SGD(0.001), #lower learning rate
              loss='categorical_crossentropy',
              metrics = ['accuracy'])

history_ = model.fit(x = train_generator,epochs = FINE_TUNING_EPOCHS ,validation_data = validation_generator)
history = history.append(pd.DataFrame(history_.history) , ignore_index=True)
```

Figure 5: SGD Classifier

For fine tuning we have utilized SGD classifier. Stochastic Gradient Descent (SGD) is an efficient way of fitting linear classifiers and regressors to complex loss functions such as Logistic Regression and Support Vector Machines (SVM). Even though SGD has been available for very large time in the machine learning sector, it has only recently been used for analyzing large scale data.

2.2.3 Epochs

```
earlyStoppingCallback = tf.keras.callbacks.EarlyStopping(monitor='val_loss',
                                                         patience=EARLY_STOPPING_CRITERIA,
                                                         verbose= 1 ,
                                                         restore_best_weights=True
                                                        )
```

Figure 6: Early Stopping

The technique of early stopping was also employed. When a monitored metric stops improving, this component of the keras library ceases training.

```
359/359 [==============================] - 79s 219ms/step - loss: 1.8105 - accuracy: 0.5991 - val_loss: 1.6117 - val_accuracy: 0.6037
Epoch 6/30
359/359 [==============================] - 79s 220ms/step - loss: 1.4504 - accuracy: 0.6312 - val_loss: 1.3862 - val_accuracy: 0.6257
Epoch 7/30
359/359 [==============================] - 79s 220ms/step - loss: 1.2335 - accuracy: 0.6567 - val_loss: 1.2698 - val_accuracy: 0.6232
Epoch 8/30
359/359 [==============================] - 79s 220ms/step - loss: 1.1022 - accuracy: 0.6755 - val_loss: 1.1925 - val_accuracy: 0.6269
Epoch 9/30
359/359 [==============================] - 79s 220ms/step - loss: 0.9911 - accuracy: 0.6989 - val_loss: 1.1901 - val_accuracy: 0.6147
Epoch 10/30
359/359 [==============================] - 79s 221ms/step - loss: 0.9158 - accuracy: 0.7182 - val_loss: 1.1777 - val_accuracy: 0.6278
Epoch 11/30
359/359 [==============================] - 79s 220ms/step - loss: 0.8419 - accuracy: 0.7416 - val_loss: 1.1470 - val_accuracy: 0.6391
Epoch 12/30
359/359 [==============================] - 79s 221ms/step - loss: 0.7778 - accuracy: 0.7650 - val_loss: 1.1937 - val_accuracy: 0.6372
Epoch 13/30
359/359 [==============================] - 80s 222ms/step - loss: 0.7305 - accuracy: 0.7792 - val_loss: 1.1541 - val_accuracy: 0.6459
Epoch 14/30
359/359 [==============================] - ETA: 0s - loss: 0.6714 - accuracy: 0.7995Restoring model weights from the end of the best epoch: 11.
359/359 [==============================] - 80s 222ms/step - loss: 0.6714 - accuracy: 0.7995 - val_loss: 1.1875 - val_accuracy: 0.6457
Epoch 14: early stopping
```

Figure 7: Epoch during training

While building the initial model, 14 epochs were run, and it stopped due to early stopping. Whereas, during fine tuning of the model, total of 20 epochs were run.

```
Epoch 8/20
359/359 [==============================] - 77s 214ms/step - loss: 0.5858 - accuracy: 0.8326 - val_loss: 1.2161 - val_accuracy: 0.6474
Epoch 9/20
359/359 [==============================] - 77s 214ms/step - loss: 0.5764 - accuracy: 0.8358 - val_loss: 1.2196 - val_accuracy: 0.6487
Epoch 10/20
359/359 [==============================] - 77s 214ms/step - loss: 0.5631 - accuracy: 0.8415 - val_loss: 1.2310 - val_accuracy: 0.6494
Epoch 11/20
359/359 [==============================] - 76s 212ms/step - loss: 0.5575 - accuracy: 0.8443 - val_loss: 1.2398 - val_accuracy: 0.6471
Epoch 12/20
359/359 [==============================] - 77s 214ms/step - loss: 0.5541 - accuracy: 0.8444 - val_loss: 1.2312 - val_accuracy: 0.6506
Epoch 13/20
359/359 [==============================] - 77s 214ms/step - loss: 0.5449 - accuracy: 0.8495 - val_loss: 1.2466 - val_accuracy: 0.6471
Epoch 14/20
359/359 [==============================] - 77s 214ms/step - loss: 0.5321 - accuracy: 0.8525 - val_loss: 1.2472 - val_accuracy: 0.6525
Epoch 15/20
359/359 [==============================] - 77s 214ms/step - loss: 0.5296 - accuracy: 0.8518 - val_loss: 1.2601 - val_accuracy: 0.6502
Epoch 16/20
359/359 [==============================] - 77s 214ms/step - loss: 0.5162 - accuracy: 0.8619 - val_loss: 1.2566 - val_accuracy: 0.6513
Epoch 17/20
359/359 [==============================] - 77s 214ms/step - loss: 0.5164 - accuracy: 0.8593 - val_loss: 1.2650 - val_accuracy: 0.6468
Epoch 18/20
359/359 [==============================] - 77s 214ms/step - loss: 0.5073 - accuracy: 0.8605 - val_loss: 1.2752 - val_accuracy: 0.6469
Epoch 19/20
359/359 [==============================] - 77s 214ms/step - loss: 0.5033 - accuracy: 0.8659 - val_loss: 1.2790 - val_accuracy: 0.6495
Epoch 20/20
359/359 [==============================] - 77s 215ms/step - loss: 0.4977 - accuracy: 0.8648 - val_loss: 1.2820 - val_accuracy: 0.6497
```

Figure 8: Epoch during Fine-tuning

2.2.4 Image of training and validation loss against iteration

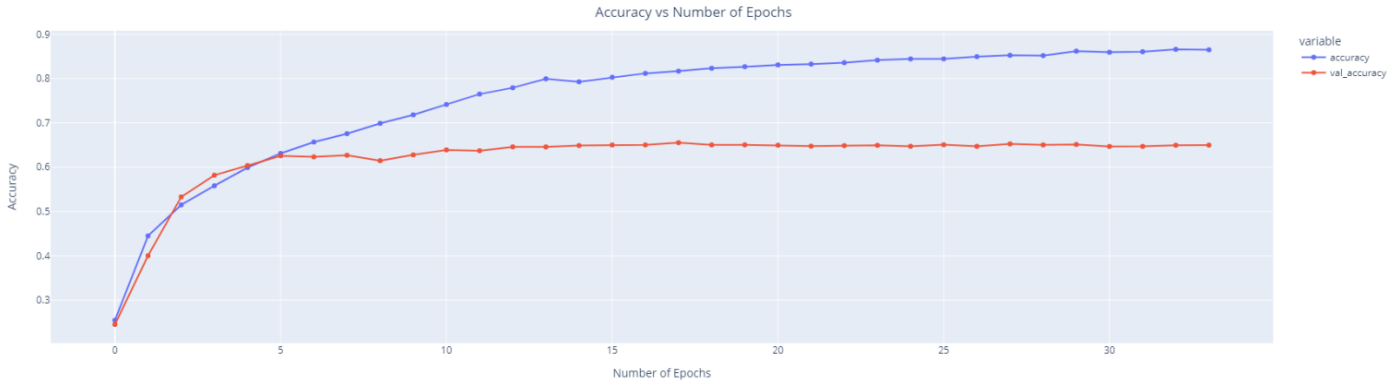


Figure 9: Accuracy Vs Number of Epochs

Here, in Figure 9, we can observe the improve in accuracy along with the increment in number of iteration of Epochs.

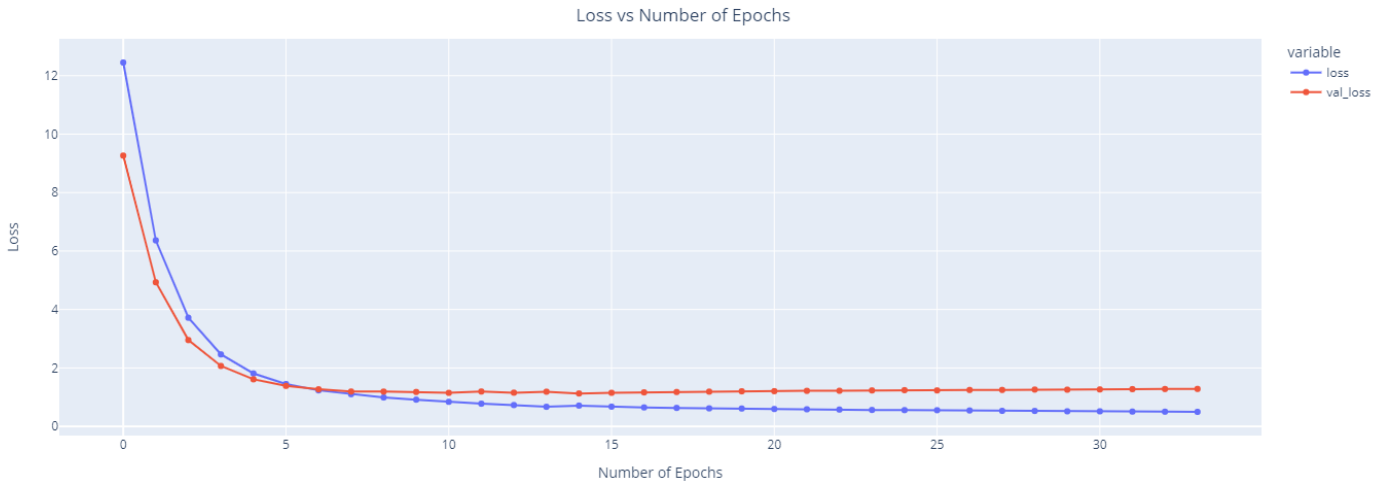


Figure 10: Loss Vs Number of Epochs

Similarly, from Figure 10, we can observe the decrease in loss along with the increase in the number of iteration of Epochs.

3. Findings and Discussion:

3.1 Evaluation metrics:

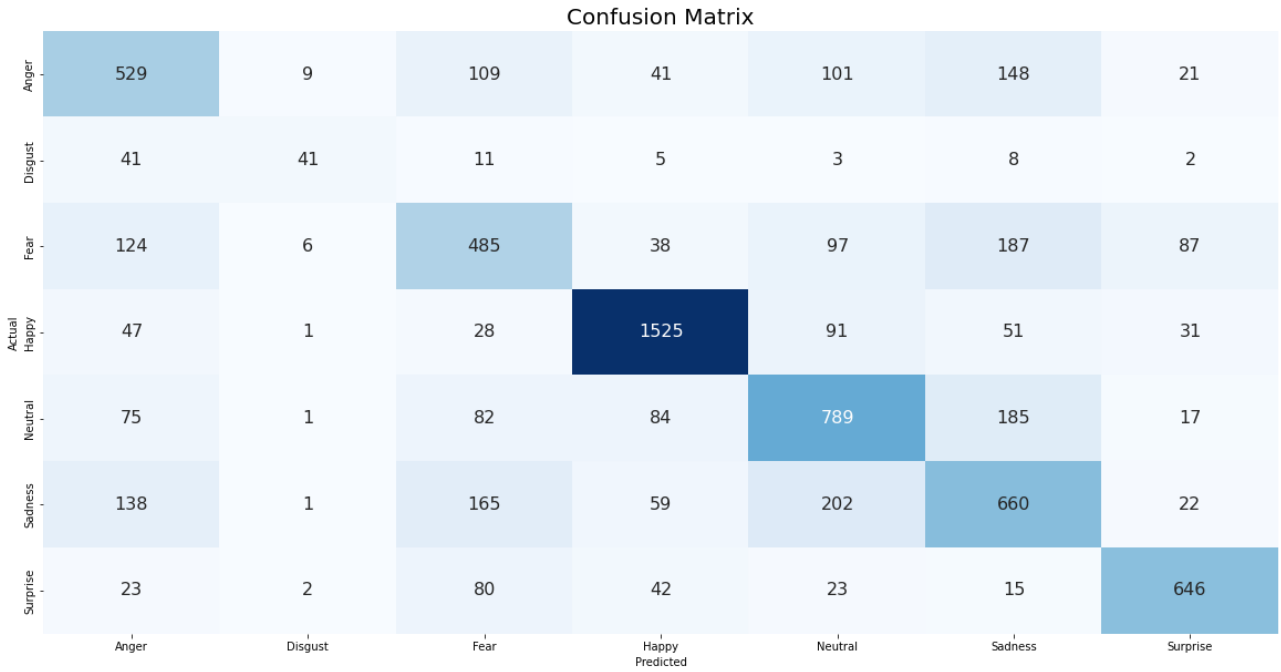


Figure 11: Confusion Matrix

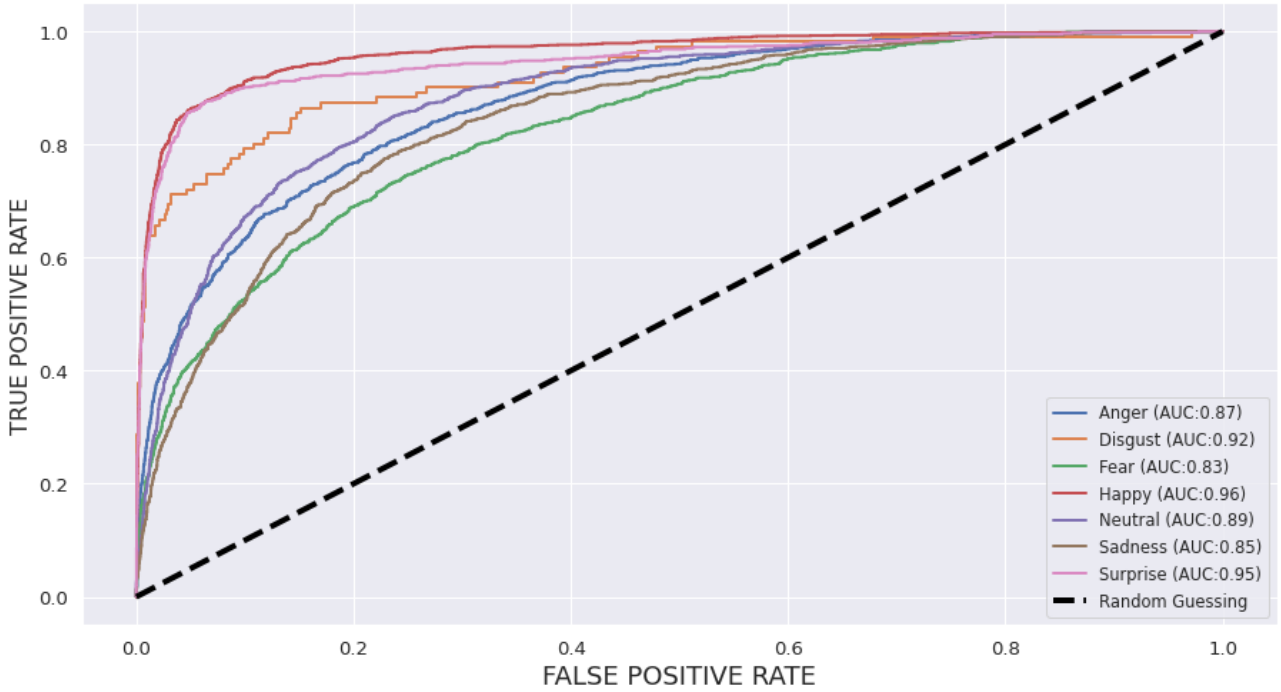


Figure 12: ROC AUC Curve

Classification Report

```
print(classification_report(y_test, y_preds))
```

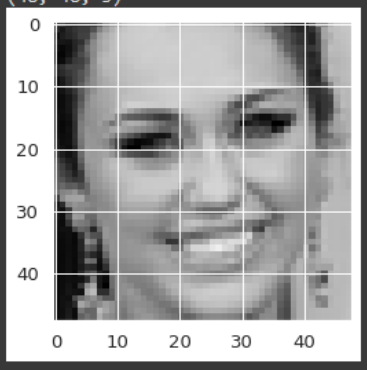
	precision	recall	f1-score	support
0	0.54	0.55	0.55	958
1	0.67	0.37	0.48	111
2	0.51	0.47	0.49	1024
3	0.85	0.86	0.85	1774
4	0.60	0.64	0.62	1233
5	0.53	0.53	0.53	1247
6	0.78	0.78	0.78	831
accuracy			0.65	7178
macro avg	0.64	0.60	0.61	7178
weighted avg	0.65	0.65	0.65	7178

Figure 13: Classification Report

```
plt.imshow(img)

print(img.shape)
```

```
(48, 48, 3)
```




```
label_dict = {0:'Angry',1:'Disgust',2:'Fear',3:'Happy',4:'Neutral',5:'Sad',6:'Surprise'}
```

```
img = np.expand_dims(img,axis = 0)
result = model.predict(img)
result = list(result[0])
print(result)
```

```
[0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0]
```

```
img_index = result.index(max(result))
print(label_dict[img_index])
plt.show()
```

```
Happy
```

Figure 14 :Test Case - Successful Classification of a Happy Face

Because the data set contains photos of poor quality and tiny size, the model's accuracy cannot be expected to be particularly good. The dataset comprises very small photos and an uneven distribution of classes, making it difficult to generate more solid findings.

References

Boesch, G., 2022. *A Complete Guide to Image Classification in 2022.* [Online]
Available at: https://viso.ai/computer-vision/image-classification/
[Accessed 29 April 2022].
Brownlee, J., 2020. *A Gentle Introduction to Cross-Entropy for Machine Learning.* [Online]
Available at: https://machinelearningmastery.com/cross-entropy-for-machine-learning/
[Accessed 28 April 2022].
Huang, G., Liu, Z., Maaten, L. v. d. & Weinberger, K. Q., 2017. Densely Connected Convolutional Networks. *CVPR 2017.*
Saha, S., 2018. *A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way.* [Online]
Available at: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53
[Accessed 28 April 2022].