Deep Learning based on Image Recognition Convolutional Neural Networks

Salah Alamri

Computer Science Department, Computing College-AlQunfuda, Umm Al-Qura University, Saudi Arabia

Abstract
Deep learning has grown in popularity over the last few decades. This technique has a variety of applications, including self-driving automobiles, successful web search, statement recognition, and image recognition. Deep learning's success gradually spreads into everyday lives. Deep learning is a sort of artificial intelligence (AI) technology that allows technology to learn independently and without being explicitly programmed. This is a fascinating and difficult matter with the potential to shape technology's future. This paper develops an image recognition system using python programming languages and the most popular deep learning workflow, Convolutional Neural Network, or CNN. We also use Keras and TensorFlow, a third-party library, to perform these operations.
Keywords: *CNN, Neural Networks, Keras, TensorFlow, deep learning.*

1.Introduction

In 1943 Walter Pitts and Warren McCulloch developed the concept of a neural network type of machine learning regarding the organization and operation of biological brain networks to replicate the processes that occur in the brain [2]. Individual components known as "neurons" comprise neural networks. Neurons are organized into clusters and layers. Each layer's neurons are all linked to neurons in follow layer. Processed molecules transfer data between both the input and output layers. In the network, each node conducts the necessary mathematical calculation sending information to every node it is linked to.

Neural networks are made up of fundamental elements that are like neurons. These units are connected and have the advantage of being modified because of a learning process or algorithm. To determine its activation state, almost all these units combine the information supplied by its connections separately (in tandem). The unit's activation is then a linear or nonlinear function of its response. In general, linear algebra ideas are utilized to examine linear units, with eigenvectors and eigenvalues being the central concepts. This investigation demonstrates the striking resemblance between linear neural networks and the general linear model constructed by statisticians [4].

Manuscript received April 5, 2022
Manuscript revised April 20, 2022
https://doi.org/10.22937/IJCSNS.2022.22.4.66

The CNN comprises many layers of neurons, each of which is a non-linear operation on a linear modification of the previous layer's outputs. Convolution layer and pooling layers are the most popular layers. Pooling layer activation is modified to use a fixed function, while convolution layers weights should be learned [12].

As a sort of artificial neural network design in 1998, Yann LeCun invented convolutional neural networks [1]. CNN makes use of some visual cortex properties. The visible critical application of this architecture is image categorization. CNN, for example, is used by Facebook for automatic tagging algorithms, Amazon for product suggestions, and Google for picture search.

2. Literature Review

Much research on Image Recognition in CCN has been published in recent years. This section provides an overview of current Image Recognition in CCN as it applies to this article. Karen Simonyan et al. [3] study how the depth of a convolutional network influences its quality in a large-scale image recognition environment, Karen Simonyan et al. [3] write. As a result, their crucial contribution is a thorough analysis of growing depth networks employing a modest (3 x 3) convolution filter construction, revealing that extending the depth to 16–19 weight layers can give improvements over earlier arrangements. Their findings formed the basis of their ImageNet Challenge 2014 entry. They won first and second overall in the localization and categorization tracks, in both, demonstrating that representations generalize very much diverse datasets and obtain state-of-the-art results. As a result, they had no choice but to make two of the better-performing ConvNet models public to aid research study on using deep pictorial representations in computer vision.

The 1.2 million categorized rising-precision images in ImageNet LSVRC-2010 competition into 1000 unique categories, Alex et al. [5] "built a vast, deep convolutional neural network." As a result, they obtained top-one and top-five mistake ratios 37.5 % and 17.0 % on test data, far better than existing state-of-the-art. The neural network has 650,000 neurons and 60 million parameters. It has 5

convolution layers. Some have been preceded by high-pooling layers, 3 totally connected layers, and a 1000-way SoftMax at the finish. To speed up training, they operated a high-efficiency and non-saturating neurons GPU version of the convolution technique. To avoid over-fitting in totally connected layers, they used a conventional regularization approach known as "dropout," which proved quite effective. Their entry of a model version won the ILSVRC-2012 contest with a top-five testing fallacy of 15.3 %, rapprochement to 26.2 % for the second-better access. They will enter in the ILSVRC-2012 contest with a variant vehicle, and as a result should take first place with a 15.3 %, rapprochement to 26.2 % for runner-up.

Providing object categorization and detection that use the cifar-10 set of data with targeted classification and prediction of aviation photos, according to Duth P. Sudharshan et al. [6], claimed that object detection from image repositories is difficult in computer vision and image processing. They used Keras with TensorFlow support to learn, test, and generate the model on a constrained computer system. The experiments show how long it takes to learn, test, and build a model in a confined computing environment. They used 60,000 images and 25 epochs to train the system, which took 722 to 760 seconds on a Tensor Flow CPU machine. After 25 generations, the training accuracy is 96 %, and the system can recognize input image's using the trained model

Medical imaging is quickly becoming one of the essential modalities for cancer diagnosis, according to Houssam BENBRAHIM et al. [7]. It lets us see inside organs in detail and any malignancies present. The area, extent, and phase tumour lesions are depicted in these photographs. Automatically classifying skin cancers based on images is a critical endeavour that can assist doctors, laboratory technologists, researchers, and laboratory technologists in making the best judgments possible. The study, deep learning, was utilized to build a sort of model for hiding tumours in images operating a CNN established on Keras and TensorFlow. The HAM10000 dataset contains 10,015 dermatoscopic images is used to test this method. The experiment's classification results show that their model has a validation set accuracy of 94.06 % and a test set accuracy of 93.93 %. The results of the investigation, their model had a classification accuracy of 93.93 % in the test set and 94.06 % in cross-validation.

Samer Hijazi et al. [8] have collaborated on this effort. In pattern recognition and image identification, convolutional neural networks (CNNs) are frequently employed due they outperformed different methods in a variety of ways. As a result, they use CNN principles to handle the challenges of a public issue give Cadence-developed performance software and algorithms that switch off computing weight and power for unpretentious drop symbol distinction ratio. In addition, they discuss using CNNs of the challenges in embedded techniques and the critical features of Tensilica® and Cadence® Visual P5 digital signal processor (DSP) for Computer Vision, software, and Imaging on which it is built. It is well-suited for CNN applications in various recognition functions and images.

3. Convolutional Neural Networks and Neural Networks

CNN's hierarchical neural networks with alternating convolutional and subsampling layers. CNNs differ in how subsampling layers and convolutional are implemented as the nets are taught [10].

Here we examine the usage of CNNs for image categorization in further depth. The primary aim of image categorization is to accept an image as input and then classify it. This is a skill that people develop from life, and image shown in figure 1 is from an elephant. A computer sees the image entirely differently:

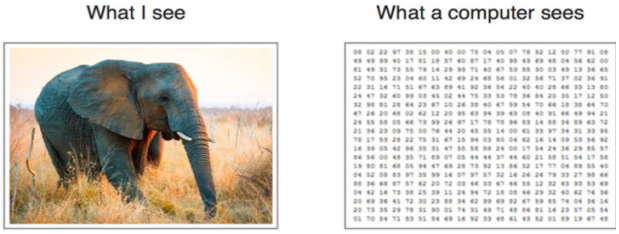

Figure 1: Human Eyes VS Computer Eyes

The computer perceives an array of pixels rather than an image. For example, 300 x 300 image size, the scenario, the array size will be 300 × 300 x 3. Where 300 is the width, three hundred is the height, and three is the RGB channel values. Each of these numbers is assigned a value ranging from 0 to 255 by the computer. This number describes the pixel's intensity at each position.

The computer looks for the base-level features to solve the recognition challenge. In human terms, such traits include the trunk and huge ears depicted in figure 1. These properties are known as borders or curvatures to computers. The computer then builds more abstract concepts using groupings of convolutional layers. Finally, before being output, the image is processed thru a set of convolutions, totally connected layers, pooling and non-linear.

3.1 Convolutional Layers

A CNN's primary building block is a convolutional layer. It is composed of a series of filters (or kernels). The characteristics of these should be taught during the learning phase. The filters are typically shorter in length than the image. To produce an activation map, each filter interacts with the image. The filter is slid across the image's width and height, and the fleck consequence between each filter element and the input is computed in each locative for convolution [11].

The picture (pixel-valued matrix) is fed into the Convolutional Layers. Assume that the input matrix reading starts just at top left of images. The software then chooses a shorter matrix there, known as a filter (or neuron or core). The filter produces convolution, and the instance advances along the input image. The filter's job is to increase its original pixel values. All these multiplications are added together to provide a single number. The filter has only read the image in upper left corner, it then moves one unit to the right at a time to complete the reading. After passing the filter through all places, a matrix is formed, which is less than the input matrix as shown figure 2.

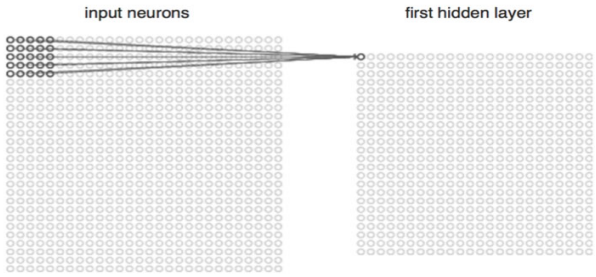

Figure 2: Convolutional Layers

This operation is akin to distinguishing visual boundaries and simple colors from a human standpoint. However, the entire network is required to recognize higher-level characteristics such as the trunk or massive ears. The network will be made up of many convolutional networks and nonlinear and pooling layers. When a picture is passed during one convolution layer, the outcome of the beginning layer becomes the inputs of the next layer. This occurs with each subsequent convolutional layer.

- Nonlinear Layer – after every convolution process, add. It has an activation function, which gives it nonlinearity. A network would be insufficiently intense and unable to model the response variable if it lacked this attribute (as a class label).
- The pooling layer comes after the non-linear layer. Operates height and width to the image and conducts a down sampling operation.

Consequently, the image's size is reduced. In addition, a few features or boundaries were recognized as in previous convolution operation, and the detailed image is no longer required for further processing and is compressed to create less complicated photos. After completing a sequence of convolutional, non-linear, and pooling layers, a fully linked layer must be attached. This layer receives data from convolutional networks. Connecting an utterly connected layer to the network's results in an N-dimensional vector, where N is several classes from which the model chooses the needed style. In the following section, we will look at a portion of the Python code for this model.

4. Performance Evaluation

This section illustrates the proposed study's performance evaluation, including data and preprocessing, VGGNet implementation, training implementation, and classification implementation.

4.1 Data and Preprocessing

The datasets are available on the instructor's website and contain around 500,000 photos. These photos depict various things that may be found in a shopping mall. Dresses, garments, electrical equipment, culinary tools, and other items are depicted. The initial stage is to perform data preprocessing, which will train a convolutional neural network. Includes recognizing and classifying each of these photos using deep learning and Keras. Surprisingly, the provided data also contains a list of pictures in comma-delimited file format, each category index. As a result, this will organize these photographs into distinct directories and label them with a category index. All of this is described in the source code below.

```
def get_images():
    images = []
    category_level1 = []
    category_level2 = []
    category_level3 = []
    count = 0
    with open('./dataset/spreadsheet/Pictures.csv') as file:
        reader = csv.reader(file)
        data = [r for r in reader]
        # add all image names to array
        for i in range(len(data)):
            images.append(data[i][0])
            category_level1.append(data[i][1])
            category_level2.append(data[i][3])
            category_level3.append(data[i][5])
            count += 1
            if count >= 250000:
                break
    return images, category_level1, category_level2, category_level3
```

Listing 1: Preprocess.py: Read CSV File and Extract Only Image Names, and Category Index (First and Second Columns)

```
def set_training_data(images, category):
    file_exist = 0
    file_not_exist = 0
    print("Create training set of " + str(len(images)) + ' images')
    for i in range(len(images)):
        image_filename = images[i]
        source = './dataset/pictures/picture/' + image_filename + '.jpg'
        destination = './training_data/' + str(category[i]) + '/' + str(
            image_filename) + '.jpg'

        if not os.path.exists('./training_data/' + str(category[i])):
            os.makedirs('./training_data/' + str(category[i]))

        try:
            #copy file from source to destination
            shutil.copyfile(source, destination)
            file_exist += 1
        except Exception:
            file_not_exist + 1
```

Listing 2: Preprocess.py: Read Images and Category Arrays and Store Specific Images in A Specific Folder

4.2. VGGNet Implementation

We use a more minor, additional consolidated variation of the VGGNet net in the CNN architecture. Zisserman and Simonyandescribed in their 2014 publication. VGGNet architectures are distinguished:

• To use 33 convolutional layers of the top in every other to increase deep.

• Using total pool to reduce quantity size.

• Layers are connected at the network's end before reaching a SoftMax classifier.

We created a minor VGGNet architecture, depicted in figure 3, to train Keras's deep learning classifier.

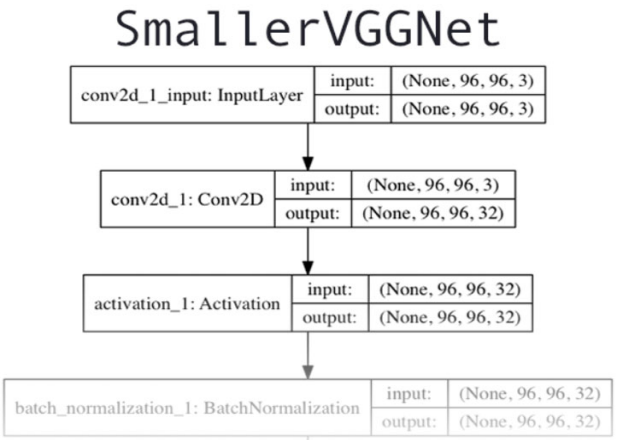

Figure 3: Smaller VGGNet

SmallerVGGNet, a scaled-down version of VGGNet, was implemented. In addition, create a new python file called smallervggnet.py and place it in the py image search folder.

Begin by importing modules as shown below:

```
from keras.models import Sequential
from keras.layers.normalization import BatchNormalization
from keras.layers.convolutional import Conv2D
from keras.layers.convolutional import MaxPooling2D
from keras.layers.core import Activation
from keras.layers.core import Flatten
from keras.layers.core import Dropout
from keras.layers.core import Dense
from keras import backend as K
```

Listing 3: smallervggnet.py: import modules

Following that, define SmallerVGGNet class as shown below:

```
class SmallerVGGNet:
  @staticmethod
  def build(width, height, depth, classes):
    # initialize the model along with the input shape to be
    # "channels last" and the channels dimension itself
    model = Sequential()
    inputShape = (height, width, depth)
    chanDim = -1

    # if we are using "channels first", update the input shape
    # and channels dimension
    if K.image_data_format() == "channels_first":
      inputShape = (depth, height, width)
      chanDim = 1
```

Listing 4: smallervggnet.py: Smaller VGGNet class

Construction strategy necessitates 4 parameters:

• Width: Width is a dimension in an image.

• Height: Image's dimension height.

• Depth: The image's depth is commonly familiar to several channels.

• Classes: Several categories in the dataset will impact the final layer in the model.

Next, add layers in begin to model as shown below:

```
    model.add(Conv2D(32, (3, 3), padding="same",
      input_shape=inputShape))
    model.add(Activation("relu"))
    model.add(BatchNormalization(axis=chanDim))
    model.add(MaxPooling2D(pool_size=(3, 3)))
    model.add(Dropout(0.25))
```

Listing 5: Convolution "CONV" => RELU =>" POOL (32 filter)

CONV => RELU => POOL block is seen above in being. Convolution layer consists of 32 filters, each of which has a 3 x 3 kernel. Following batch normalization, the activation function RELU is employed.

POOL size 3 x 3 used in the POOL layer to swiftly decrease the spatial dimensions of 96 x96 to 32 x 32 (as in the following section, train network with 96x 96 x 3 input images). Dropout will also be employed in network architecture, as you can see from the code block-dropout tasks by detaching nodes during the current layer at connecting and random them to the following layer. The use of disconnects accidental during learning collections aids in the natural introduction of profusion in the system. Because no one node into layer is reliable for indicating a particular category, item, corner, and edge, randomized disconnects during training batches aid in the natural insertion of indifference into the model. Add two levels of (CONV => RELU) before adding another POOL layer as shown below:

```
    model.add(Conv2D(64, (3, 3), padding="same",
      input_shape=inputShape))
    model.add(Activation("relu"))
    model.add(BatchNormalization(axis=chanDim))
    model.add(Conv2D(64, (3, 3), padding="same",
      input_shape=inputShape))
    model.add(Activation("relu"))
    model.add(BatchNormalization(axis=chanDim))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))
```

Listing 6: (CONV => RELU) * 2 => POOL (64 filter)

Learn a more diversified set of features by stacking many CONV and RELU layers together (before lowering that volume's spatial dimensions). Filters size is being increased from 32 to 64; the more minor the spatial dimensions of volume, the deeper the network, and the more filters learn. To avoid diminishing spatial dimensions rapidly, reduced the maximum size pooling from (3 x 3 to 2 x 2). At this point, dropout is performed once more.

Next, add other set of (CONV =>RELU) ∗2=> P OOL as shown below:

```
    model.add(Conv2D(128, (3, 3), padding="same",
      input_shape=inputShape))
    model.add(Activation("relu"))
    model.add(BatchNormalization(axis=chanDim))
    model.add(Conv2D(128, (3, 3), padding="same",
      input_shape=inputShape))
    model.add(Activation("relu"))
    model.add(BatchNormalization(axis=chanDim))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))
```

Listing 7: (CONV => RELU) * 2 => POOL (128 filter)

Raised the size of filter to 128 here. To reduce overfitting once more, 25% of the nodes are dropped.

Collection of F C => RELU layers as well as a SoftMax classifier as shown below:

```
    # first (and only) set of FC => RELU layers
    model.add(Flatten())
    model.add(Dense(1024))
    model.add(Activation("relu"))
    model.add(BatchNormalization())
    model.add(Dropout(0.5))

    # softmax classifier
    model.add(Dense(classes))
    model.add(Activation("softmax"))
    # return the constructed network architecture
    return model
```

Listing 8: FC => RELU

Dense (1024) specifies the fully linked layer with linear corrected unit batch normalization and activation. Next, dropout is done once more, seeing a drop out of fifty % of nodes thru preparing this time. In entirely linked layers, you frequently operate a dropout rate from 40-50%, and in previous levels, you use a considerably lower rate, typically 10-25%. Finally, finish the model with a SoftMax classifier, which yields anticipated probabilities for every class label.

4.3. Training Implementation

First, parse command line arguments as follows as shown below:

```
ap = argparse.ArgumentParser()
ap.add_argument("-d", "--dataset", required=True,
  help="path to input dataset (i.e., directory of images)")
ap.add_argument("-m", "--model", required=True,
  help="path to output model")
ap.add_argument("-l", "--labelbin", required=True,
  help="path to output label binarizer")
ap.add_argument("-p", "--plot", type=str, default="plot.png",
  help="path to output accuracy/loss plot")
args = vars(ap.parse_args())
```

Listing 9: Command line arguments

Three command line arguments are necessary for training script:

• –dataset: The input dataset's path.

Dataset is structured in dataset guide, with every category represented by a sub guide. In addition, category photographs can be found within each sub guide.

• –model: Output model's path — The training script trains the example and saves the sample to disk.

• –labelbin: Output label binarized path extracts the class labels. Where dataset guide terms and generates tag binarized. There is additionally one option statement.

• –plot. If you establish a path/filename, a plot.png file is created of your essential tasking guide.

After handled command line arguments, initialize the following variables:

(1) EPOCHS: The whole numeral of epochs for which net trained (i.e., how multiple periods net "sees" every training example and realizes patterns of it).

(2) INIT LR: The first learning ratio — 1e 3 defaulting values for the Adam improver train the net.

(3) BS: Send batches of photos on the net for a train. Each epoch has many batches. The BS value determines batch size. (4) IMAGE DIMS: Provide locative measurements of input photos here. Input photos must be 96 by 96 pixels of three channels (i.e., RGB). Smaller VGGNet was created explicitly of image. Develop information and tags to store the preprocessed pictures and tags.

Then loop through photographs and use the Keras function to convert and resize them to fit model. The data array is then converted to a NumPy array, and the pixel intensity is scaled to an area [0, 1]. Also change the tags from a list to a NumPy array. Finally, an information note is a press display size (in MB) of the information matrix. A frequent technique of deep learning, or any machine learning for an issue, is to part test and train.

Continue by generating an 80/20 random division of information. Because of the limited number of information points, 250 images per category use information enlargement throughout the procession to give additional standard images to train with (established on current photos). Information enlargement is a device each profound learn practitioner should have in their toolbox. Start Keras CNN model with spatial input dimensions of (96 × 96 x 3). The smaller VGGNet was designed to accept (96 × 96 x 3) photos as input. To use alternative spatial measurements to decrease the network deep for smaller photos and improve the network deep for bigger ones. Because we have more than two classes, utilize the Adam optimizer with knowledge ratios decline and construct a standard with categorical cross entropy. To train the network, we invoke the Keras fit-generator method.

4.4. Classification Implementation

Require label binaries and models in memory to classify the image as shown below.

```
# load the trained convolutional neural network and the label
# binarizer
print("[INFO] loading network...")
model = load_model(args["model"])
lb = pickle.loads(open(args["labelbin"], "rb").read())
# classify the input image
print("[INFO] classifying image...")
proba = model.predict(image)[0]
idx = np.argmax(proba)
label = lb.classes_[idx]
```

Listing 10: Label Binaries and Model in Memory

The image is then classified, and the label is created as shown below.

```
filename = args["image"][args["image"].rfind(os.path.sep) + 1:]
correct = "correct" if filename.rfind(label) != -1 else "incorrect"
# build the label and draw the label on the image
label = "{}: {:.2f}% ({})".format(label, proba[idx] * 100, correct)
output = imutils.resize(output, width=400)
cv2.putText(output, label, (10, 25),  cv2.FONT_HERSHEY_SIMPLEX,
  0.7, (0, 255, 0), 2)
# show the output image
print("[INFO] {}".format(label))
cv2.imshow("Output", output)
cv2.waitKey(0)
```

Listing 11: Identify Correctness

The name of each category index is extracted from the filename and compared to the tag. The appropriate variable is "incorrect" to "correct." These two lines presume that your receiving image does have a filename that includes the tag.

Following that, proceed as follows:

•To the class label, add the likelihood % and "correct" / "incorrect" wording. • Scale the generated image to fit screen.

• On the resulting image, draw the label text.

• Show the output image while waiting for a keypress to leave.

5.Result

The results of the performance evaluation are presented in this section.

5.1 Training Result

Run the subsequent command to train the method, being foolproof to supply the command line options correctly as shown below:

```
$ python train.py --dataset dataset --model product.model --labelbin lb.pickle
Using TensorFlow backend.
[INFO] loading images...
[INFO] compiling model...
[INFO] training network...
Epoch 1/100
29/29 [==============================] - 2s - loss: 1.4015 - acc: 0.6088 -
    val_loss: 1.8745 - val_acc: 0.2134
Epoch 2/100
29/29 [==============================] - 1s - loss: 0.8578 - acc: 0.7285 -
    val_loss: 1.4539 - val_acc: 0.2971
Epoch 3/100
29/29 [==============================] - 1s - loss: 0.7370 - acc: 0.7809 -
    val_loss: 2.5955 - val_acc: 0.2008
...
Epoch 98/100
29/29 [==============================] - 1s - loss: 0.0833 - acc: 0.9702 -
    val_loss: 0.2064 - val_acc: 0.9540
Epoch 99/100
29/29 [==============================] - 1s - loss: 0.0678 - acc: 0.9727 -
    val_loss: 0.2299 - val_acc: 0.9456
Epoch 100/100
29/29 [==============================] - 1s - loss: 0.0890 - acc: 0.9684 -
    val_loss: 0.1955 - val_acc: 0.9707
[INFO] serializing network...
[INFO] serializing label binarizer...
```

Listing 12: Training Process

According to the result of the train script, Keras CNN achieved 96.84% sort precision on the train set. The test set, 97.07 % precision. Figure 4 shows that train the standard for 100 epochs and obtained minimal loss limit overfitting. However, improve accuracy by collecting more training data.

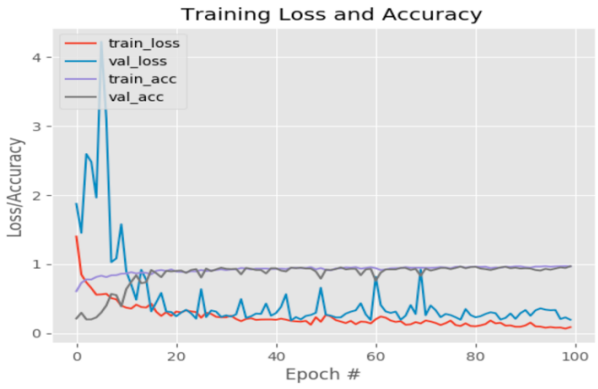

Figure 4: Training and Validation Loss/Accuracy Charts

5.2. Classification Results

Classification Results. (Big: Correct, Small: Incorrect) as shown in figure 5.

Figure 5: Classification Results. (Big: Correct, Small: Incorrect)

6. Discussion

VGGNet is unique in this study. It is "sometimes referred to VGG," according to this understanding of VGGNet. Simonyan, as well as Zisserman, established it in their 2014 paper [3]. Their main contribution was to show that architecture with tiny (33) filters could be trained to increase depths (16-19 layers) and realize state-of-the-art classification on challenging ImageNet classification assignment.

6.1 Smaller VGGNet: Going Deeper with CNNs

Deep learning network topologies formerly consumed a variety from filter sizes.

Filter lengths in the CNN's first layer generally range between 7x7 [5] to 11x11 [9]. As a result, filter sizes were gradually decreased to 55, with only the network's innermost tiers using 33 filters in the end.

VGGNet is notable for its use of 33 seeds during the design. Using tiny seeds presumably allows VGGNet to generalize to sort problems unrelated to the context in which it was trained.

If you come across a network architecture that is only made up of 33 filters, you can be sure it was influenced by VGGNet. The full VGGNet 16- and 19-layer varieties are too forward to introduce Convolutional Neural Networks.

We looked at the VGG network family to see what characteristics a CNN must have to be recognized as a member of this family, and then built a smaller VGGNet that can be easily trained on system.

6.2 The VGG Family of Networks

Two fundamental characteristics separate the VGG family of Convolutional Neural Networks.

In the network's CONV layers, only 33 filters are employed.
Before conducting a POOL operation, stack several CONV => RELU layer sets (with the amount of sequential CONV => RELU layers rising as more profound).

The use of a VGGNet variant that is substantially shallower, known as ‘smaller VGGNet.

6.3 The (Smaller) VGGNet Architecture

This employs a chain from CONV => RELU => POOL layers in both ShallowNet and LeNet. However,

with VGGNet, several CONV => RELU layers are layered before a solo POOL layer is applied. This enables the network to learn more detailed information of the Convolution layer before down - sampling the spatial input size with the POOL function. The smaller VGGNet is made up of 2 series from CONV => RELU => CONV => RELU => POOL layers, then a series from FC => RELU => FC => SOFTMAX levels.

7. Conclusion and FUTURE WORK

The little training data is one of the model's critical shortcomings. We tried on various photos, and the classifications were sometimes inaccurate. When this occurred, we studied input picture + network further and observed that the color visible in the image considerably influences sort, for instance, if an image contains a lot of reds and oranges. This is due in part to our input data. Because these generated images are blatantly fake, there are no actual "real-world" photos of them. We were inspired by fan art or movie/TV program stills for our images.
Furthermore, we only had a limited amount of data (225-250 images). When training a Convolutional Neural Network, we should have at least 500-1,000 images for every class. Remember this when working with your data.

We conducted an experiment using the dataset. In the future, we will increase the amount of data and strive to improve the quality of the results to extend and improve accuracy.

Acknowledgment
I want to convey heartfelt gratitude to Dr Jin, Ruoming for his invaluable guidance.

References

[1] Yann Le Cunand Yoshua Bengio. "The handbook of brain theory and neural networks". Chapter Convolutional Networks for Images, Speech, and Time Series, pages 255–258. MIT Press, Cambridge, MA, USA, 1998.

[2] Warren S. McCulloch and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity". *The bulletin of mathematical biophysics*, 5(4):115–133, Dec 1943.

[3] Karen Simonyan and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition". *CoRR*, abs/1409.1556, 2014.

[4] Herve ABDI. "A neural network primer". Journal of Biological Systems, 1994, 2.03: 247-281.

[5] KRIZHEVSKY, Alex; SUTSKEVER, Ilya; HINTON, Geoffrey E. “Imagenet classification with deep convolutional neural networks”. Advances in neural information processing systems, 2012, 25: 1097-1105.

[6] Duth P. SUDHARSHAN, Swathi RAJ. “Object recognition in images using convolutional neural network”. In: 2018 2nd International Conference on Inventive Systems and Control (ICISC). IEEE, 2018. p. 718-722.

[7] Houssam BENBRAHIM, Hanaâ HACHIMI, Aouatif AMINE. “Deep Convolutional Neural Network with TensorFlow and Keras to Classify Skin Cancer Images”. Scalable Computing: Practice and Experience, 2020, 21.3: 379-390.

[8] Samer Hijazi, Rishi Kumar, and Chris Rowen, IP Group, Cadence. “Using convolutional neural networks for image recognition”. Cadence Design Systems Inc.: San Jose, CA, USA, 2015, 1-12.

[9] Pierre SERMANET, et al. “Overfeat: Integrated recognition, localization and detection using convolutional networks”. arXiv preprint arXiv:1312.6229, 2013.

[10] Dan C. Cires¸an, Ueli Meier, Jonathan Masci, Luca M. Gambardella, Jurgen Schmidhuber. “Flexible, high performance convolutional neural networks for image classification”. In: Twenty-second international joint conference on artificial intelligence. 2011.

[11] Sakib MOSTAFA, Fang-Xiang WU. “Diagnosis of autism spectrum disorder with convolutional autoencoder and structural MRI images.” In: Neural Engineering Techniques for Autism Spectrum Disorder. Academic Press, 2021. p. 23-38

[12] Qiuhong, KE, JunLiu, Mohammed Bennamoun, SenjianAn,FerdousSohel,Farid Boussaid, . “Computer vision for human–machine interaction”. In: Computer Vision for Assistive Healthcare. Academic Press, 2018. p. 127-145.

Salah Alamri received his B.S. degree in Computer Science from King Abdelaziz University, Saudi Arabia in 2010, and received M.S. degree in Computer Science from Kent State University, USA in 2014. He received the PhD degree in Computer Science in August 2020 from Kent State University, USA. He is currently work as an assistant professor, Computer Science department in the Faculty of Computing at Al-Qunfudhah branch at Umm Al-Qura University.