Weather Classifying Convolutional Neural Network

Machine Learning Final Project Report

ENEL 525 - Machine Learning for Engineers
Professor Henry Leung
December 17, 2021

Dan Tran
UCID: 30050533

Table of Contents

List of Figures

List of Tables

1 Introduction

This project involved creating a neural network which can classify images according to weather conditions. This neural network will utilize deep-learning and backpropagation as its main corrective algorithm. The convolutional neural network (CNN) will be the type of neural network primarily considered for the principal architecture for this network. A multi-class weather dataset (MWD) of weather images was used to train this network.

The goal of this project was to develop image processing and classification workflow techniques, evaluate and assess model performance, and understand the use of common and popular machine learning libraries Tensorflow, OpenCV, and numpy [1]. These techniques were learned in class and online tutorials, then applied directly to the neural network.

This report describes the preprocessing procedure and architecture of the final neural network, then gives the output values from training and testing. The testing data and personal data outputs are analyzed in detail to review the network performance. Other possible architectures for the CNN are tested and analyzed as well to determine improvements to the neural network that could improve its performance.

2 Preprocessing and Network Architecture

The neural network can be split cleanly into two portions: preprocessing and architecture. Section 3.1 shall describe the preprocessing while section 3.2 will describe the architecture.

2.1 Preprocessing

There are 4 major steps to data preprocessing, each labelled A, B, C, and D. A defines classes and creates input variables, B populates the target array, C preprocesses individual images and populates the image array, and D randomizes the data. The preprocessing portion of the code is shown in the figure below.

```
# image preprocessing and setting targets
# A: Formatting
path = "D:\Python\Weather_Project\dataset2"
rimg_list = []
targets = []
class_names = ['cloudy', 'rain', 'shine', 'sunrise']
inputwidth = 64

for filename in os.listdir(path):
    # B: target code
    for idx, val in enumerate(class_names):
        if(filename.find(val) != -1):
            targets.append(idx)
            break

    # C: read file -> correct colour -> resize
    img = cv2.imread(os.path.join(path,filename))
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)              # correct colour -> no greyscale
    rimg = cv2.resize(img, (inputwidth,inputwidth)) / 255.0 # normalize as well as resize
    rimg_list.append(rimg)

rimg_list = np.array(rimg_list)
targets = np.array(targets)

# Set a seed for repeatability
np.random.seed(seed=0)

# D: Split data (80%, 20%)
permuted_index = np.random.permutation(rimg_list.shape[0])
index_80 = (int)(rimg_list.shape[0] * 0.80)
X_train = rimg_list[permuted_index[0:index_80]]
Y_train = targets[permuted_index[0:index_80]]
X_test = rimg_list[permuted_index[index_80+1:]]
Y_test = targets[permuted_index[index_80+1:]]
```

Figure 1: Preprocessing Portion of the Code

2.1.1 Point A - Defining Classes and Input Width

Step A defines the input arrays “rimg_list” and “targets”. “rimg_list” is the array that will contain processed images while “targets” holds each image’s respective target. The four classes the images will be classified into are defined in the “class_names” array as “cloudy,” “rain,”

"shine," and "sunrise." These classes are not defined by the neural network developer, but rather defined in a dataset used by a research paper entitled "Multi-class weather recognition from still image using heterogeneous ensemble method." [2]

2.1.2 Point B - Import Each File and Classify its Target

Step B imports individual image files in a loop. The filenames in the dataset have their classes in them ("cloudy1.jpg", "shine109.jpg", etc.). This property is used to determine the target of each image, then insert this information into the "targets" array.

2.1.3 Point C - Image Preprocessing Steps

Step C reads the image to a variable, and performs the following preprocessing steps:

1. Correct the colour from BGR to RGB
2. Resize the image to 64x64 pixels
3. Normalize the channel values

For colour, since cv2 reads images as BGR, rather than the correct RGB order, the images must be corrected. In addition, there was the option of removing colour and grey-scaling the images as well as filtering for lines. However, due to the lack of overall shape in weather-related images, the choice was made to maintain colour. For instance, sunrise and shine images tend to only be differentiable by colour, as the overall features tend to be similar. Maintaining colour would more accurately separate the two classes.

To maintain speed and reliability, all images were resized to 64x64 pixels. This way, defining features in smaller and larger images may be detected more readily by the same convolutional layers. Channel values were additionally normalized to maintain consistency with all images. It additionally prevents the convolutional layers from handling large channel values by restricting the values to [0,1].

2.1.4 Point D - Data Randomization

The final process of data processing is randomizing the data. The images are randomized to ensure that the neural network is inputted with a correct proportion of images from all four classes. Due to cv2 functionality, all images are read alphabetically, which means that images are not randomized. Without randomization, the network will primarily be trained on cloudy and rain images. This will severely reduce accuracy since the network was not trained on shine and sunrise images.

Additionally, as required by project requirements, the data is split into 80% and 20% portions for training and testing, respectively. The final training arrays are "X_train" and "Y_train" while the testing arrays are "X_test" and "Y_test."

2.2 Network Architecture

2.2.1 Convolutional Neural Network

Convolutional neural networks were chosen as the primary architecture for this project due to its design focus and strengths around image classification [3]. CNNs use kernels to detect "features" from an image that could correspond to a classification of weather. For instance, "shine" images will typically have blue skies and a sun in them. CNNs can detect these features regardless of their location in the image, given that the kernels were correctly trained to detect those features. Thus, due to these unique designs of CNNs, they were chosen to classify the images.

2.2.3 Final Layer Parameters

The final architecture of the neural network is shown below in Figure 3.2

```
# E: Create machine learning architecture
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(inputwidth, inputwidth, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(4))

model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

Figure 2: Final CNN Architecture

The final architecture developed was two, main convolution layers with 3x3 kernels and "relu" activation functions. The first, 32-neuron layer has a pooling layer to compile its data. The latter 64-neuron convolution layer sends its data to a "flatten" layer, then to a 64-neuron dense layer. The last layer is a 4-neuron dense layer, each representing one of the defined sorting classes.

The convolution layers are built from kernels which are calibrated to certain features present on each class. The max pooling layer compiles the features' presence into numbers which later layers use to determine the presence of certain features.

All layers use the "relu" activation function due to its high compatibility with convolution. Convolution uses the presence of features to classify images. The presence of these features must be given in full value, which is achieved with the "relu" activation function which does not cut off at 1. In contrast, the sigmoid activation function contains the output in the range: [0,1]. Because of this limited range, the presence of certain recognizable features cannot be

emphasized when using the sigmoid (“softmax” in code) activation function. Both of these functions are displayed below in Figure 3.

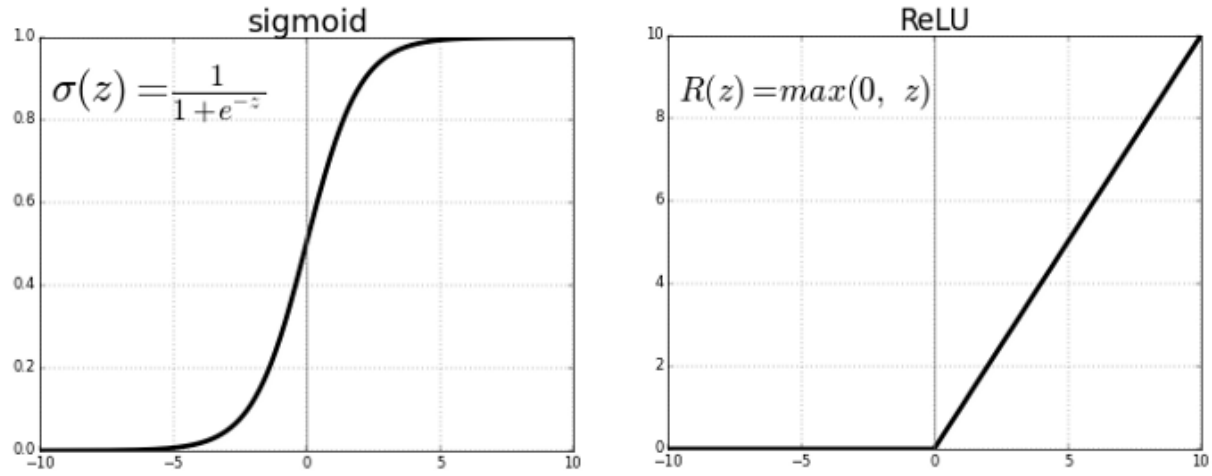


Figure 3: Sigmoid and ReLU activation functions [4]

The neural network uses the ‘adam’ optimizer, and determines loss using Sparse Categorical Crossentropy during the compilation process. The resulting architecture can be seen in Figure 4 below.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 62, 62, 32)        896

 max_pooling2d (MaxPooling2D  (None, 31, 31, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 29, 29, 64)        18496

 flatten (Flatten)           (None, 53824)             0

 dense (Dense)               (None, 64)                3444800

 dense_1 (Dense)             (None, 4)                 260

=================================================================
Total params: 3,464,452
Trainable params: 3,464,452
Non-trainable params: 0
```

Figure 4: model.summary() output

2.2.4 Other Layers and Kernel Sizes Tested

Other layer combinations and kernel sizes were tested, and their resulting outputs analyzed. These other architectures will be discussed in more detail in section 4.3 of this report.

2.3 Training Parameters

Through testing, the optimum number of epochs to be inputted into the neural network is 20. This is shown below in Figure 4.

```
# Train the model
history = model.fit(X_train, Y_train, epochs=20, validation_data=(X_train, Y_train))
```

Figure 5: Code for Model Training

3 Results

3.1 Model Outputs

For the model described in Section 3 of the report, the outputs are displayed in the following table:

Table 1: Model Outputs

Output Parameter	**Value**
Training Set Accuracy	1.0000
Testing Set Accuracy	0.9152
Loss	0.2702
Epoch Time	478 ms
Step Time	68 ms

After 20 epochs, the accuracy of the model, with respect to the training set, is 100%. However, the accuracy on the testing set is 91.52%. The loss is 0.2702, and epoch training time is 0.478 seconds. Figure 6 shows the accuracy and val_accuracy values over the course of the 20 training epochs. The neural network achieves above 90% accuracy within 5 epochs, with fine adjustments over the remaining epochs.

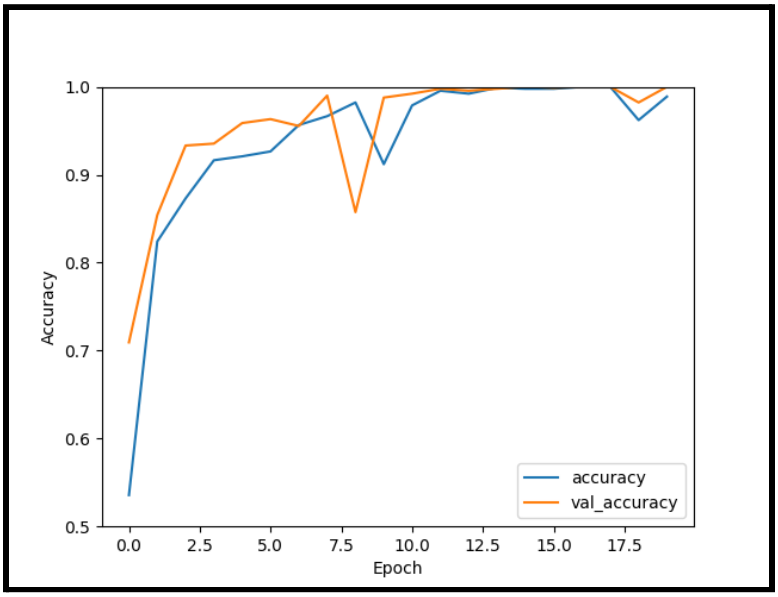


Figure 6: Accuracy Graph for the Neural Network

The following table displays the confusion matrix for the architecture. As shown, there are more mismatches between “cloudy” and “shine” images, whereas “rain” and “sunrise” images tend to be classified more correctly.

Table 2: Confusion Matrix for the Neural Network

	cloudy	**rain**	**shine**	**sunrise**
cloudy	53	7	4	0
rain	2	40	0	1
shine	3	4	37	0
sunrise	0	0	0	73

3.2 Sample Images

10 personal, sample images were used to test the model on real-life images not found within the testing dataset. They are shown in Figure 7 below:

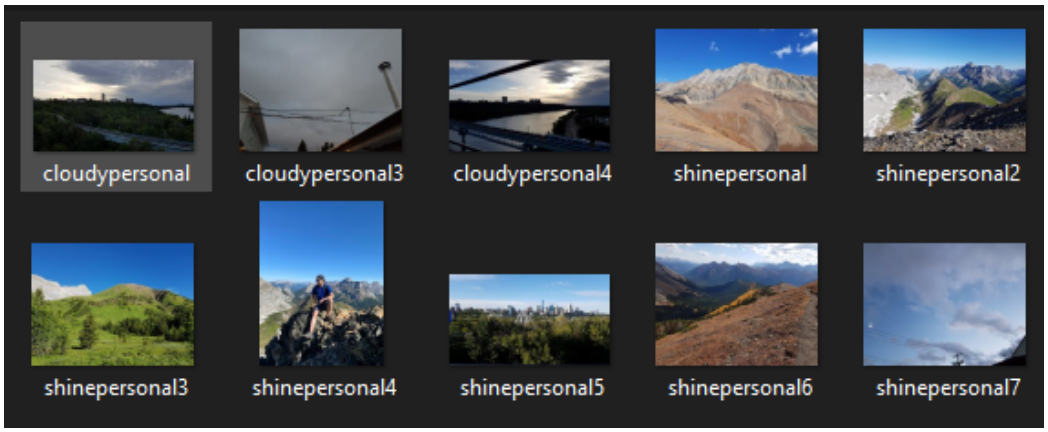


Figure 7: Personal Test Images

These images were processed using the same preprocessing steps as defined in Section 3.1. The output figure below shows the results of feeding these images through the neural network.

```
cloudypersonal.jpg  is predicted as:  cloudy
shinepersonal.jpg  is predicted as:  sunrise
shinepersonal2.jpg  is predicted as:  shine
shinepersonal3.jpg  is predicted as:  shine
shinepersonal7.jpg  is predicted as:  shine
shinepersonal4.jpg  is predicted as:  shine
cloudypersonal3.jpg  is predicted as:  cloudy
shinepersonal6.jpg  is predicted as:  rain
cloudypersonal4.jpg  is predicted as:  sunrise
shinepersonal5.jpg  is predicted as:  shine
1/1 - 0s - loss: 2.4992 - accuracy: 0.7000 - 20ms/epoch - 20ms/step
Personal photos accuracy =  0.699999988079071
```

Figure 8: Personal Test Image Results

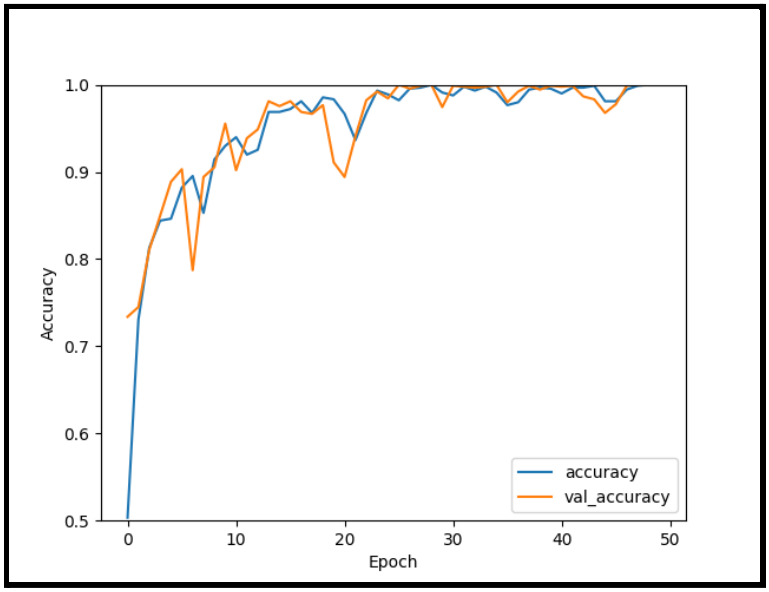
As clearly shown, the accuracy of the model on these personal images was only 70%, with a loss of 2.4992.

3.3 Alternate Model Tests

Other architectures were tested for training time and accuracy throughout the development of the final architecture. All of these architectures were tested over 50 epochs to guarantee that they reach 100% accuracy on the training data. They are then tested on the test data from the MWD.

Each of these architectures are labelled after their number of neurons in each convolution layer. For instance, Architecture 32-64-64 indicates that this architecture has 3 convolution layers, the first with 32 neurons, the second with 64 neurons, and the last with 64 neurons.

The corresponding accuracy graphs and confusion matrices of all of the tested CNN architectures are shown below in Figures 9-13:




```
The Confusion Matrix is:
 [[57.  1.  6.  0.]
 [ 6. 36.  1.  0.]
 [ 3.  1. 40.  0.]
 [ 0.  0.  1. 72.]]
```

Figure 9: Performance for Architecture 32-64-64

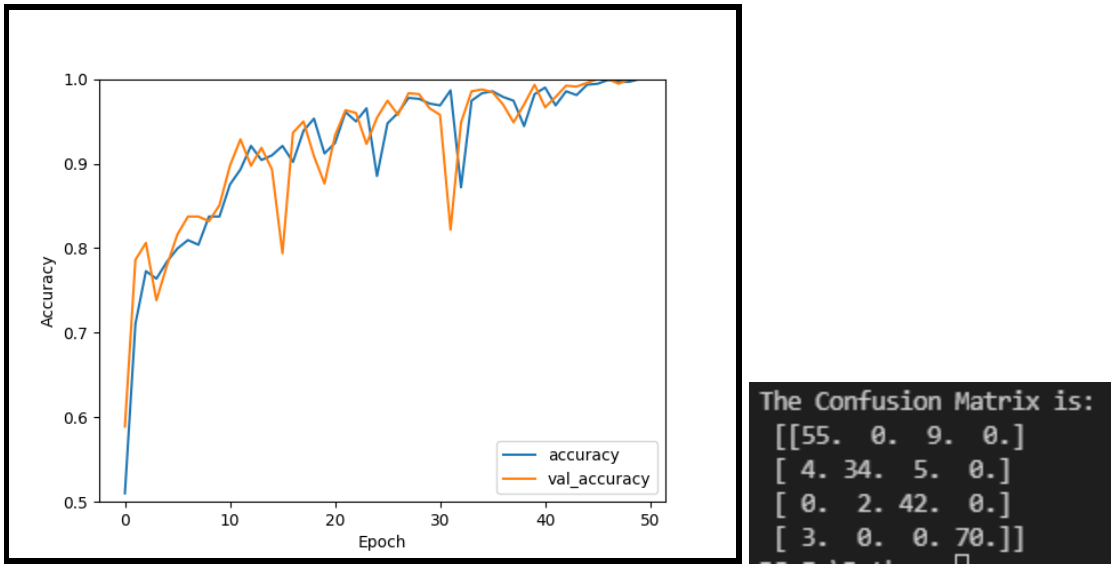


Figure 10: Performance for Architecture 32-32-64-64

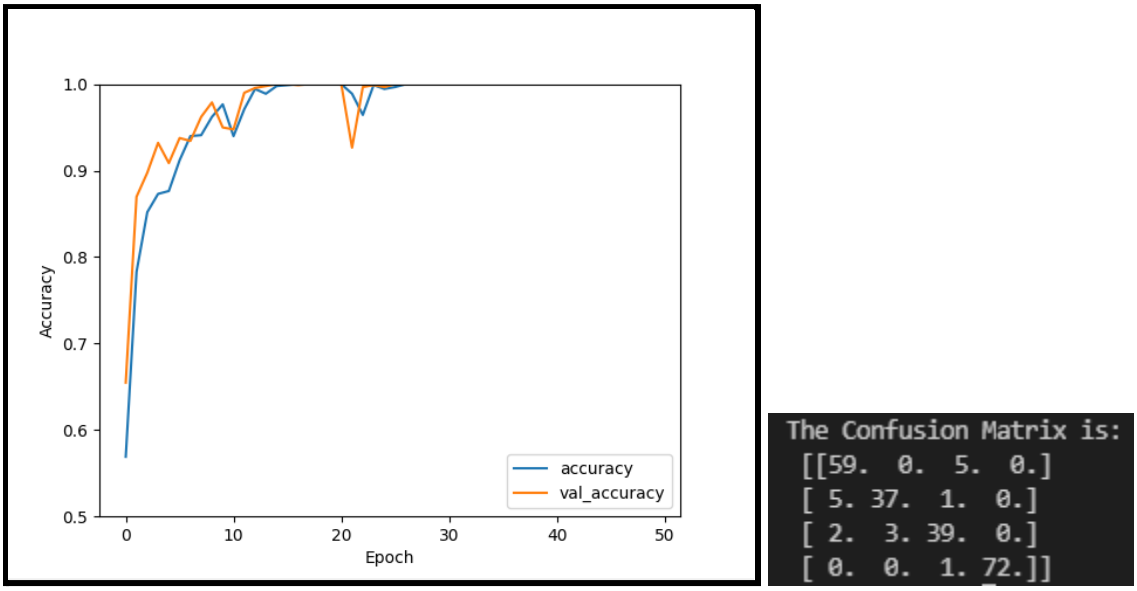
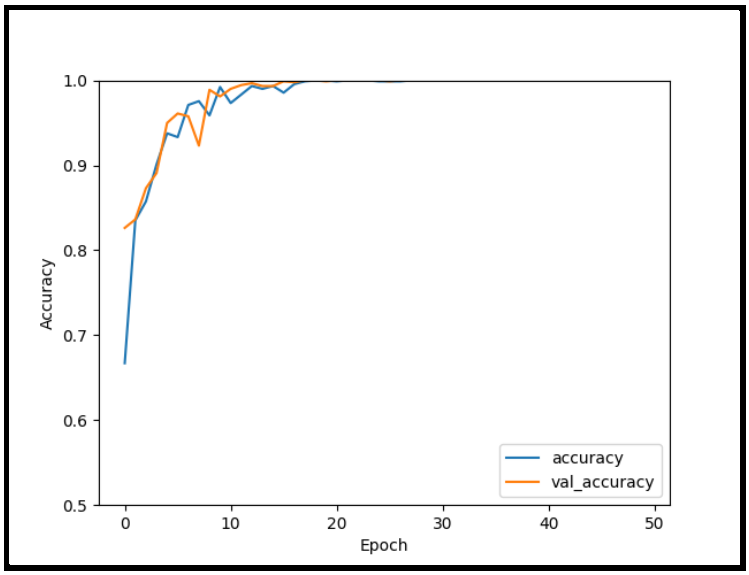
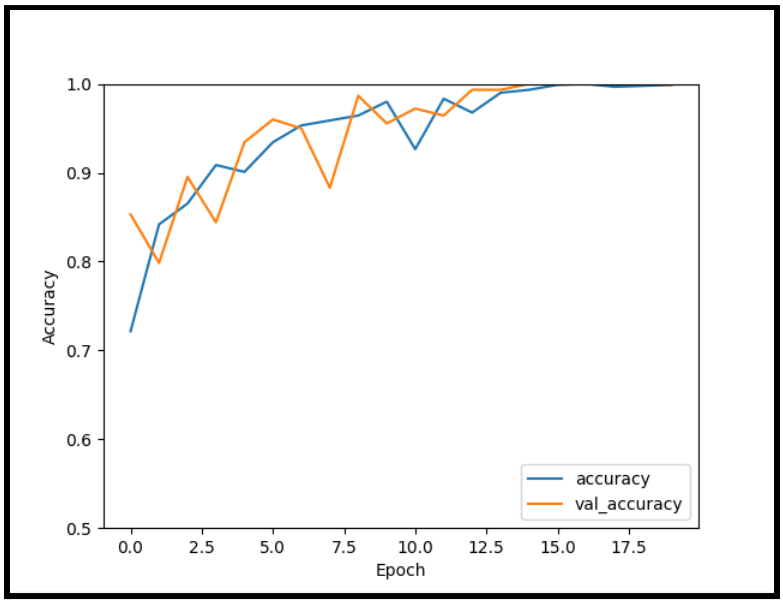


Figure 11: Performance for Architecture 32-64




```
The Confusion Matrix is:
 [[57.  1.  5.  1.]
 [ 7. 34.  1.  1.]
 [ 2.  2. 40.  0.]
 [ 0.  0.  1. 72.]]
```

Figure 12: Performance for Architecture 64




```
The Confusion Matrix is:
 [[58.  0.  6.  0.]
 [ 5. 36.  1.  1.]
 [ 1.  2. 41.  0.]
 [ 0.  0.  1. 72.]]
```

Figure 13: Performance for Architecture 32-64 (5x5 kernels)

These performance results, using accuracy of test-image classification, and number of epochs to 100% accuracy can be compiled into the following table:

Table 3: Test Results of other CNN Architectures

Architecture	Accuracy (%)	Epochs to 100% Training Accuracy
32-64-64*	92.56	29
32-32-64-64	93.30	46
32-64**	93.75	15
64	91.52	18
32-64 (5x5 kernels)	92.86	19

**Read as: 32-neuron convolution layer, then 64-neuron convolution layer, then 64-neuron convolution layer CNN network*
***The final network architecture chosen*

As shown by the table above, all of these architectures have an accuracy in the 90-94% range, but with significantly varying number of epochs to train to 100% accuracy with the training dataset.

4 Discussion

4.1 Analysis

4.1.1 Weather Dataset

Using the Multi-class weather dataset (MWD), the accuracy for the 32-64 architecture and 20 training epochs was 91.52%. This accuracy is reasonably high for a 2-layer CNN. The neural network additionally trains to 100% on the testing data very fast at about 15 epochs.

However, there is a discrepancy between the 91.52% accuracy on the testing dataset, and the 100% of the training dataset. This can be attributed to over-training, where the model is fitted too closely to the training dataset. Limiting the number of epochs could be a correction for this, as more generalized networks could classify new data more readily, rather than being too calibrated on the testing data. However, this ideal number of epochs is very difficult to determine, since a neural network can rarely achieve 100% accuracy with new data unless it was able to train on it.

Additionally, according to the confusion matrix shown in Table 2, cloudy and shine images are misclassified the most, with 11 and 7 misclassifications respectively. Rain and sunrise images are misclassified 3 and 0 times respectively. This can be attributed to the unique features of these images. Sunrise images are the most unique with red/orange colours, which enable the neural network to classify these images easily. Rain images tend to be unique in terms of features as well, since no other image classes contain raindrops. However, cloudy and shine images have significant overlap of features with the other classes, which cause them to be mis-classified more readily.

4.1.2 Personal Images

The neural network performs with 70% accuracy (out of a test of 10 images) when using personal images. This is far below the expected accuracy of 91.52% from the multi-class weather dataset.

This can be attributed to a lack of feature overlap between the personal images and dataset images. For instance, many of the shine images in the dataset have the sun, which is not present in the personal images. The personal images are more vague than the dataset images, and therefore do not have the crisp features expected by the CNN.

Additionally, there are no sunrise or rain images due to a lack of them available for testing. To improve the results from testing personal photos, choosing images that match the feature-set of the testing images will increase accuracy.

4.1.4 Determining the Optimum CNN Architecture

When training each of the different architectures on 50 epochs, the 32-64 architecture trained the fastest at 15 epochs. This is compared to the 29 epochs of the 32-64-64 architecture, and 46 epochs of the 32-32-64-64 architecture.

Due to the simplicity of weather images, and their general lack of distinct features, the simpler 32-64 architecture trained to 100% accuracy the fastest. In comparison with the larger 32-64-64 and 32-32-64-64 architectures which took longer to train with little to no performance gain. Due to this, it can be assumed that at a certain point, the architecture of the neural network can be heavily reduced in size without significant loss to the testing accuracy. With the reduction comes a massive performance gain that the larger networks cannot replicate.

However, too large of a reduction can decrease the accuracy of the neural network. A single layer of 64 convolution neurons demonstrated a slight performance decrease compared to the larger networks. Because of this, the medium-sized architecture of 32 and 64 convolution neurons was deemed to be the optimal architecture for this project.

4.2 Areas for Improvement

4.2.1 Improving the Dataset

Looking at the images in the MWD, many of the images are not completely weather-related. In the following figure are 5 images from the dataset that exemplify this:

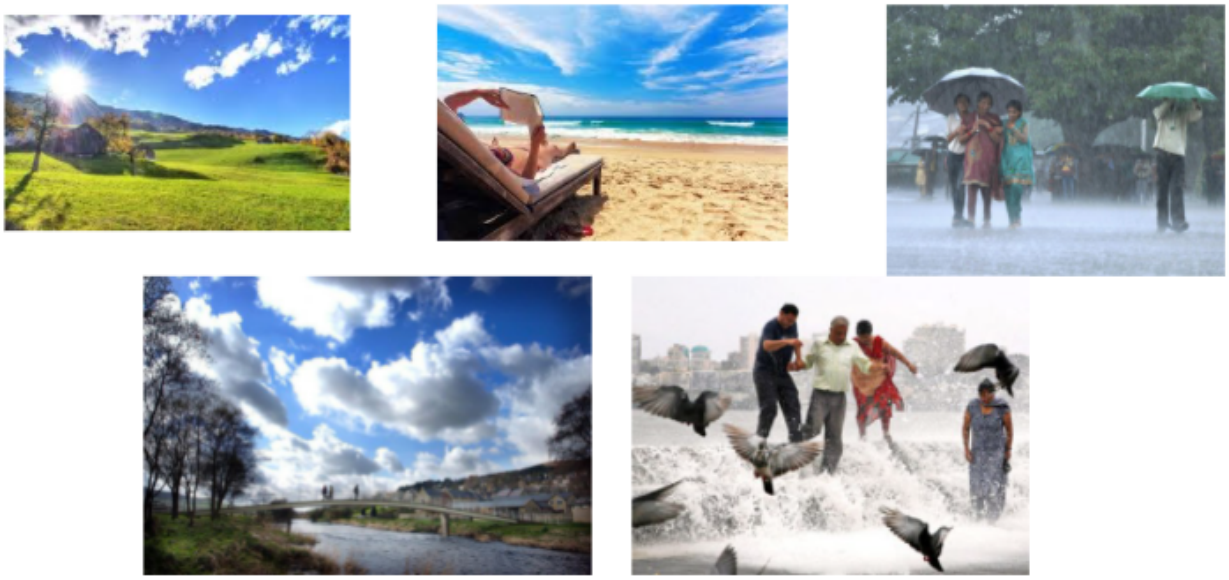


Figure 14: Impure Image Samples

As can be seen from these five images, they have contaminants. Non-weather objects such as people, animals, terrain, water elements, and flora are present which may incorrectly bias the convolution layers towards detecting features not associated with weather.

The required goal of the project should be defined clearly: is the objective of the neural network to classify pure weather images (a weather camera pointing directly up at the sky), or is the objective to classify random outdoor photos? If the goal is the former, selecting a massive dataset of uncontaminated, pure weather images will push accuracy as close as possible to 100%. If the latter is the goal, then using this dataset will be adequate, however the accuracy will be difficult to increase beyond ~93% without more data.

Another improvement to increasing accuracy would be to augment the existing images (through rotation, reflection, etc.) to artificially multiply the current dataset multiple times over. This could potentially allow the convolution neurons to truly train on features present on these images, no matter their location [5].

4.2.2 Use of Fuzzy Logic

An additional improvement that may improve accuracy would be to implement fuzzy logic. This would help due to some overlap of the weather images. This overlap is shown in the following figure:

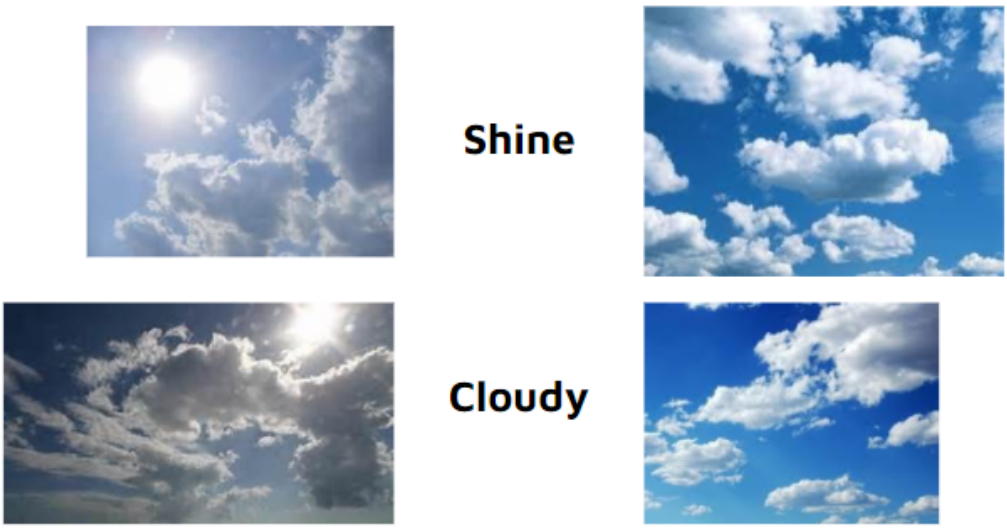


Figure 15: Images with Overlapping Features

In the above figure, the top two images are classified as shine images, while the bottom two are classified as cloudy images. However, if visually compared, the two images on the right side are more similar to each other, and the images on the left side are more similar to each other than their classifications.

Subjectively, all four of these images may be classified as being both shine and cloudy simultaneously. This creates the issue where all four may be classified crisply as shine or cloudy incorrectly. Fuzzy logic would account for this overlap in features properly, and possibly classify the images more correctly.

However, changing the architecture to a fuzzy-logic CNN would require re-classifying all of the dataset images manually as well as reworking the outputs (confusion matrix, loss, accuracy, etc.) to handle the fuzzy nature. The scope of this project was determined to not require any significantly higher accuracy or speed so fuzzy logic was exempted.

5 Conclusion

In conclusion, this project demonstrates the utility of CNNs in classifying weather-related images into the correct classes. The CNN, with a good architecture, can achieve up to 94% accuracy while maintaining a high training speed. This project also demonstrates the capability of the Tensorflow library in machine learning; the CNN architecture was built easily and seamlessly in Python with limited training.

Because of the few and limited features of weather, a CNN with two convolution layers worked extremely well, performing on par with larger networks while having a significantly smaller training period. The classification of personal images, however, was relatively poor. This was most likely caused by the choice of a poor set of personal test images.

To truly push towards the 100% accuracy, there are additional options for improvements. The first is to scrutinize and standardize all of the images to ensure that they do not contain contaminating features, and the other is to utilize fuzzy logic to encapsulate the non-discrete nature of weather. However, even without these significant modifications, the CNN still demonstrated a very high accuracy and learning speed.

6 References

[1] Leung, H. (2021, November), “ENEL 525 Project Report/Presentation,” [Online]. Available: https://d2l.ucalgary.ca/d2l/le/content/400184/viewContent/5026116/View [Accessed December 17, 2021].

[2] Oluwafemi, A. G., & Zenghui, W. A. N. G. (2019, January). Multi-Class Weather Classification from Still Image Using Said Ensemble Method. In *2019 Southern African Universities Power Engineering Conference/Robotics and Mechatronics/Pattern Recognition Association of South Africa* (SAUPEC/RobMech/PRASA) (pp. 135-140). IEEE.

[3] S. Saha, “A comprehensive guide to Convolutional Neural Networks - the eli5 way,” Medium, 17-Dec-2018. [Online]. Available: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53. [Accessed: 18-Dec-2021].

[4] S. Sharma, “Activation functions in neural networks,” Medium, 04-Jul-2021. [Online]. Available: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6. [Accessed: 18-Dec-2021].

[5] T. Dayanand, “Data preprocessing and network building in CNN,” Medium, 24-Aug-2020. [Online]. Available: https://towardsdatascience.com/data-preprocessing-and-network-building-in-cnn-15624ef3a28b. [Accessed: 18-Dec-2021].