University of Calgary

ENEL 525 Fall 2023 – Final Project

Harrison Mondragon

30088805

December 18, 2023

Introduction

Introduction Summary

The project involves implementing a deep learning technique called convolutional neural networks (CNNs) for image classification. The dataset provided contains 5 classes of different flower species, and the focus of the network is to identify and recognize these various flower species using images to train, validate, and test. The primary goal is to leverage CNNs to process and learn from floral image data, exploring the intricacies of feature extraction. The project aims to achieve high accuracy in distinguishing between different floral categories, emphasizing the application of deep learning for robust and precise classification based on visual characteristics.

Methodology

Problem Set-up

I use Python and Tesorflow to achieve the goals of this project. First, I load in the dataset using the Python os module and some looping logic to collect all the image paths and their associated classes. The dataset is organized as it was on the Kaggle link, where the dataset directory contains subdirectories with the class names, containing the associated images. My code depends on this organization to import the data correctly. Next, I shuffle the data so that the training, validation, and testing sets will contain randomized data from all classes, and split the dataset using the 0.70/0.15/0.15 split outlined in the project description. Then, I utilize the ImageDataGenerator API from Tensorflow Keras to preprocess the images by normalizing and resizing them. I do this separately for training, validation, and testing so that I can manage them separately later. Next, I create and compile the CNN model using Tensorflow Keras. At this point, the data and model are ready to be used. I train the models using the train and validation sets, then evaluate the model on the test set, and output relevant plots and statistics. As I was developing my code, I used reduced versions of the dataset to incrementally test changes. I mainly developed using a dataset with 100 images per class and 5 epochs so that I could iterate quickly. However, the rest of this report will contain outputs of the full dataset with 10 epochs.

Network Design

I closely followed the network design in the final project slides to construct my model and it is included below. The only difference between the diagram and my model is that I only have 2 fully connected layers, whereas the diagram has 4.

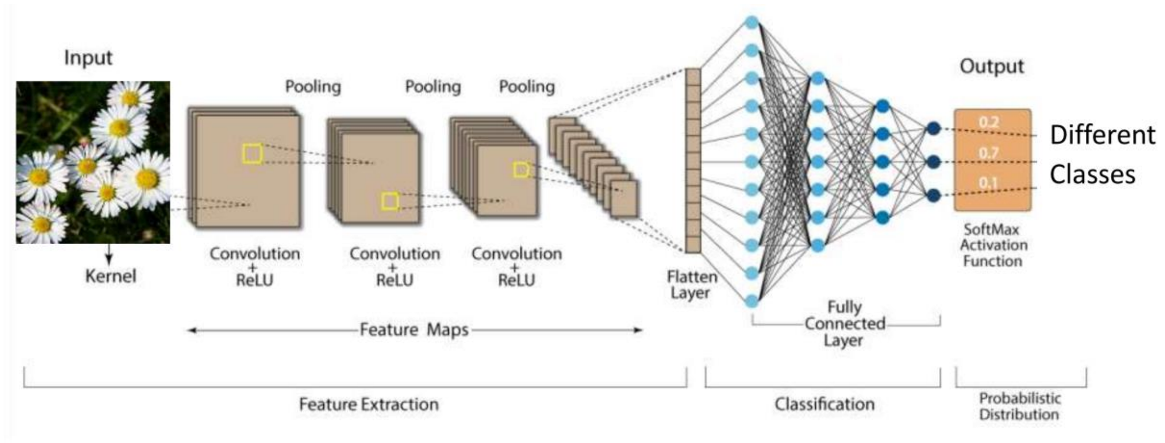


Fig 1. CNN Network [6]

Layer Design Choices (in order):

- Conv2D(32, (3, 3), activation='relu', input_shape=input_shape)
    - First convolution layer.
    - I provide the input shape that I resized to during preprocessing. It is 320x240 pixels with 3 color channels for RGB based off the project description.
    - I used 32 filters here to start off with a relatively small value, since this layer captures basic features like edges and simple textures. This helps to extract lower-level features and reduce computational load in the initial layers.
- MaxPooling2D((2, 2))
    - First pooling layer.
    - Reduces the spatial dimensions of the input by taking the maximum value in the pool size region.
- Conv2D(64, (3, 3), activation='relu')
    - Second convolution layer.
    - I used 64 filters here to gradually increase the number of filters. This helps the network learn more complex and abstract features based on the features already captured by the first layer.
- MaxPooling2D((2, 2))
    - Second pooling layer.
    - Reduces the spatial dimensions of the input by taking the maximum value in the pool size region.
- Conv2D(128, (3, 3), activation='relu')
    - Third convolution layer.
    - I used 128 filters here to gradually increase the number of filters even further. This helps the network learn more complex and abstract features based on the features already captured by the first and second layer.
- MaxPooling2D((2, 2))
    - Third pooling layer.

    - Reduces the spatial dimensions of the input by taking the maximum value in the pool size region.
- Flatten()
    - Flatten layer.
    - Transforms the output from the 3D tensor in the convolution and pooling layers into a 1D tensor vector that can be used by the densely connected layers.
- Dense(256, activation='relu')
    - First dense (fully connected) layer.
    - Bottleneck layer that consolidates all the high-level features extracted from the convolution layers before making a final decision.
    - I used 256 units here to have enough resources to capture all the complex and intricate features extracted in earlier layers.
- Dropout(0.5)
    - Dropout hyperparameter layer.
    - Drops 50% of the units during training.
    - Helps provide generalization and prevent overfitting, but causes training to be slower.
- Dense(len(class_names), activation='softmax')
    - Dense (fully connected) output layer.
    - The number of units here matches the number of classes because it produces a probability distribution for each class.
    - I used the softmax activation function to facilitate this probability distribution, and the highest probability value is chosen as the predicted class.

General Design Choices:

- I used 3 convolution layers to match the diagram. This also gave me enough layers to gradually increase the number of filters so that I could capture sufficient features from the dataset images.
- I used 3 pooling layers because it needs to match the number of convolutional layers to work together correctly.
- I used 3x3 convolutional kernels in my convolutional layers because it is a common choice for CNNs. It has a relatively small receptive field to look at a small local region of the image to capture features and details from. Stacking small kernels together like this makes the network more expressive, allowing it to capture complex patterns without increasing the number of parameters significantly.
- I used a 2x2 pool size in my pooling layers because it is a common choice for CNNs. It helps reduce spatial dimensions and computational load. I chose to go small here because pool sizes that are too large can result in loss of information.

- I used the Rectified Linear Unit (ReLU) activation function in all my layers before the output to introduce non-linearity to my model. This allows my network to learn complex patterns and relationships in the data.

CNN Backpropagation Concept

Backpropagation is an essential algorithm for training neural networks. Its main purpose is to adjust network parameters like weights and bias to improve performance over training iterations or epochs. In the context of convolutional neural networks, the backpropagation concept is used to compute the loss gradient with respect to the previous layer. This helps the network learn how much each layer's output contributed to the overall loss [5].

Essentially the data goes through a forward pass of the full network, calculates loss, then does a backwards pass all the way to the beginning of the network to calculate the loss gradient at each layer. This loss is then utilized to modify the network parameters more intelligently for further iterations, making the network more efficient and useful.

Loss Function Derivation

I am applying the categorical cross-entropy loss function to my model. It is effective for classification tasks because works with the softmax function I use for the output layer of my model to produce a multi-class classification output. Tensorflow has functionality to use this loss function selection as a parameter when compiling a model, the derivation of the actual formula is included below. In the formula, C is the number of classes, y_i is the true probability distribution vector (it will be one-hot encoded for my program), and s_i is the predicted probability vector (which will be the output from my final output layer in my model). When using this loss function to minimize loss during training, the probability distribution of the softmax output layer are intended to eventually resemble the one-hot encoded true vectors[4].

$$\mathcal{L}(\boldsymbol{y}, \boldsymbol{s}) = -\sum_{i=1}^{C} y_i \cdot log(s_i)$$

Fig 2. Categorical Cross-Entropy Function Derivation [4].

Training Scheme

For the training scheme percentages, I followed the specifications in the project outline to use 70% for training and 15% for validation. I train the model using the keras model fit command by passing the train generator and validation generators I created earlier in the code, and 10 epochs. As with the testing scheme, I randomized the full dataset before making the set splits, so every time the code is ran, different images will be used for training and validation. I used the adam optimizer to get a stochastic gradient descent method based on adaptive estimation of first order and second order moments as I learned from project lesson 2. I used categorical crossentropy as my loss function as I learned from project lesson 2.

Testing Scheme

I followed the specification in the project outline to use 15% of the dataset for testing. I did not use any images from outside the dataset for testing. While developing my code, this 15% would have been from a smaller subset of the full dataset (100 images per class or 300 images per class), but I used the full dataset for the results in this report. Every time the code is ran, the dataset order is randomized, so the actual images used for testing will be unique per run, but it will always be 15% of the total.

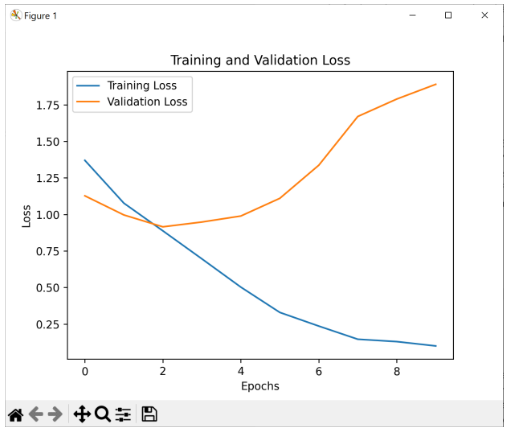
Results and Discussion

Relevant Parameters and Training Curves

Relevant Parameter Table:

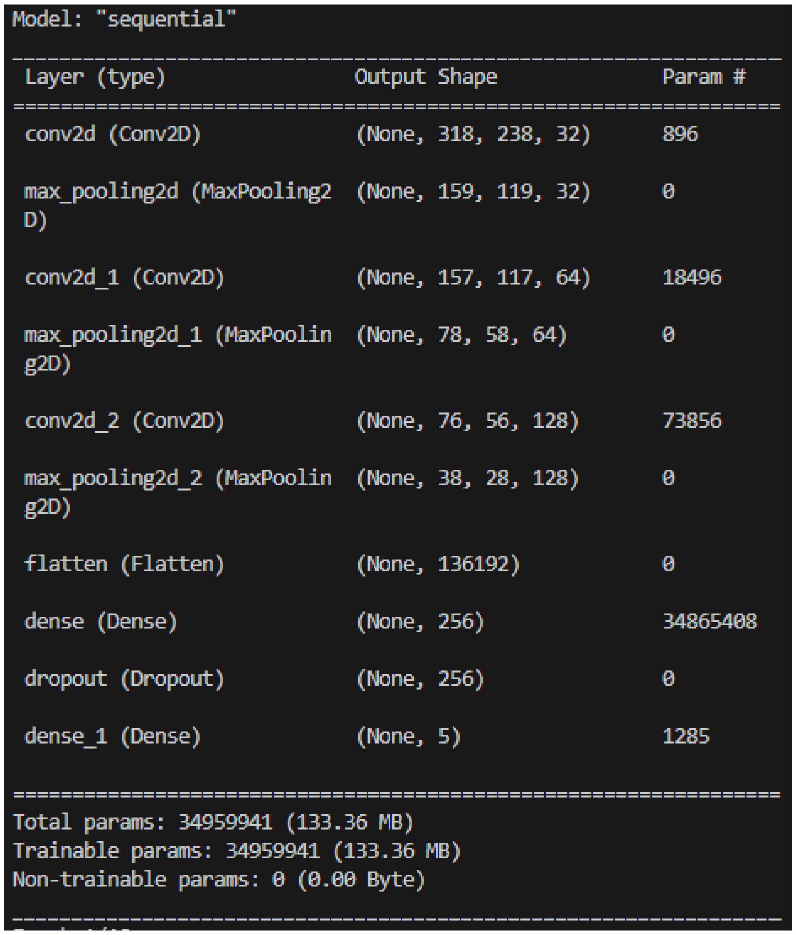
Learning Rate	Did not explicitly specify in code. Defaults to **0.001**
Optimizer	**adam**
Loss Function	**categorical_crossentropy**
Number of epochs	**10**

Training and Validation Loss Curves:



Model Summary and Training Parameters

Model summary console output:



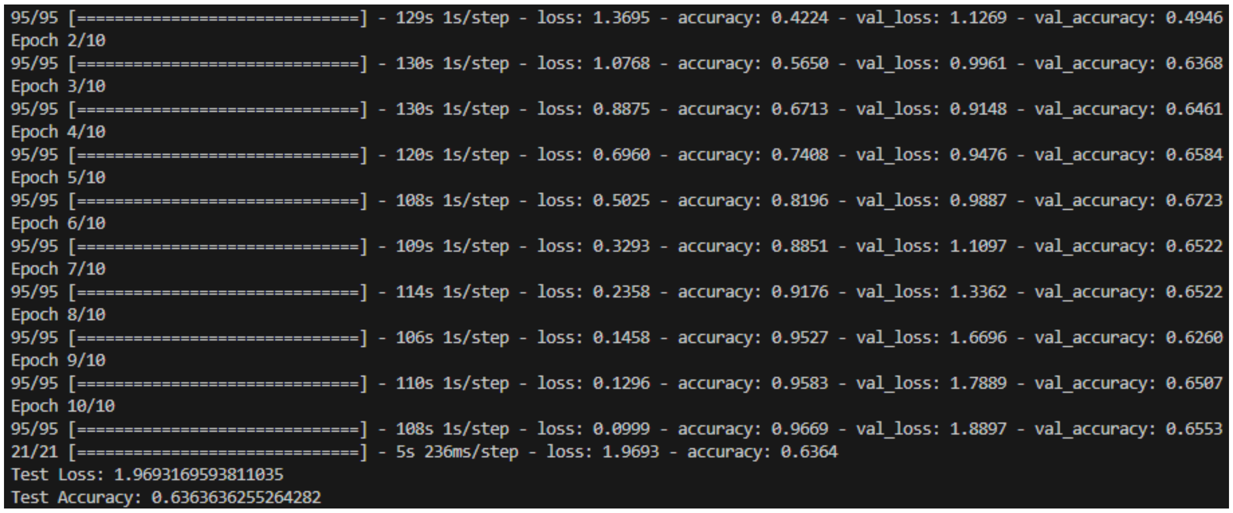
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 318, 238, 32)      896

 max_pooling2d (MaxPooling2  (None, 159, 119, 32)      0
 D)

 conv2d_1 (Conv2D)           (None, 157, 117, 64)      18496

 max_pooling2d_1 (MaxPoolin  (None, 78, 58, 64)        0
 g2D)

 conv2d_2 (Conv2D)           (None, 76, 56, 128)       73856

 max_pooling2d_2 (MaxPoolin  (None, 38, 28, 128)       0
 g2D)

 flatten (Flatten)           (None, 136192)            0

 dense (Dense)               (None, 256)               34865408

 dropout (Dropout)           (None, 256)               0

 dense_1 (Dense)             (None, 5)                 1285

=================================================================
Total params: 34959941 (133.36 MB)
Trainable params: 34959941 (133.36 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

I have 3 convolution layers with 3 pooling layers to perform feature extraction. The filters of the convolution layers increase from 32 to 64 to 128. All convolution layers are Conv2D with 3x3 kernels and a ReLU activation function. All pooling layers are MaxPooling2D with 2x2 pool sizes. I have 2 fully connected (dense) layers after a Flatten layer, with a Dropout layer in between. The first dense layer has 256 units, and the second dense layer has as many units as there are classes (5 with the flower dataset). The first dense layer has a ReLU activation function, and the last one has a softmax activation function.

Final Results

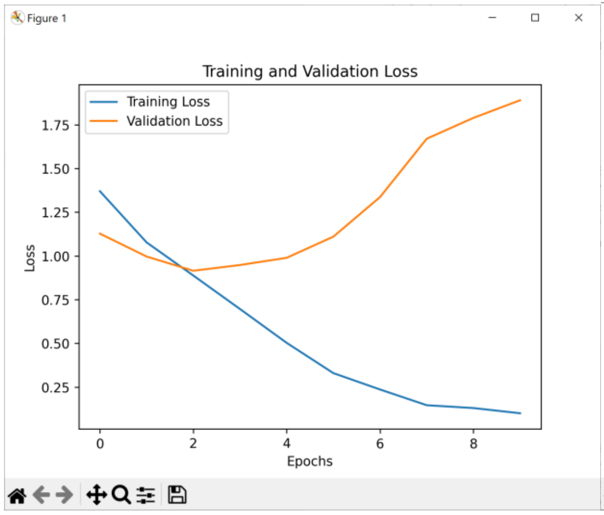
The final results vary run to run, but the below image shows a relatively standard output. The training accuracy reaches 96.7% with 0.01 loss by the last epoch. The validation accuracy reaches 65.5% with 1.89 loss by the last epoch. The testing accuracy is 63.6% with a 1.97 loss.

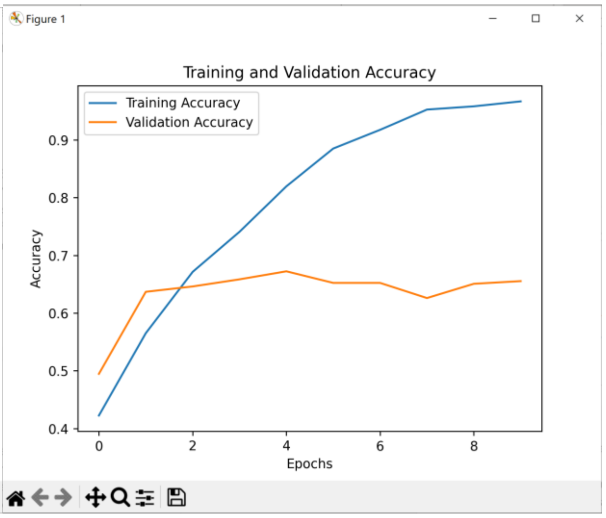
Final results console output:



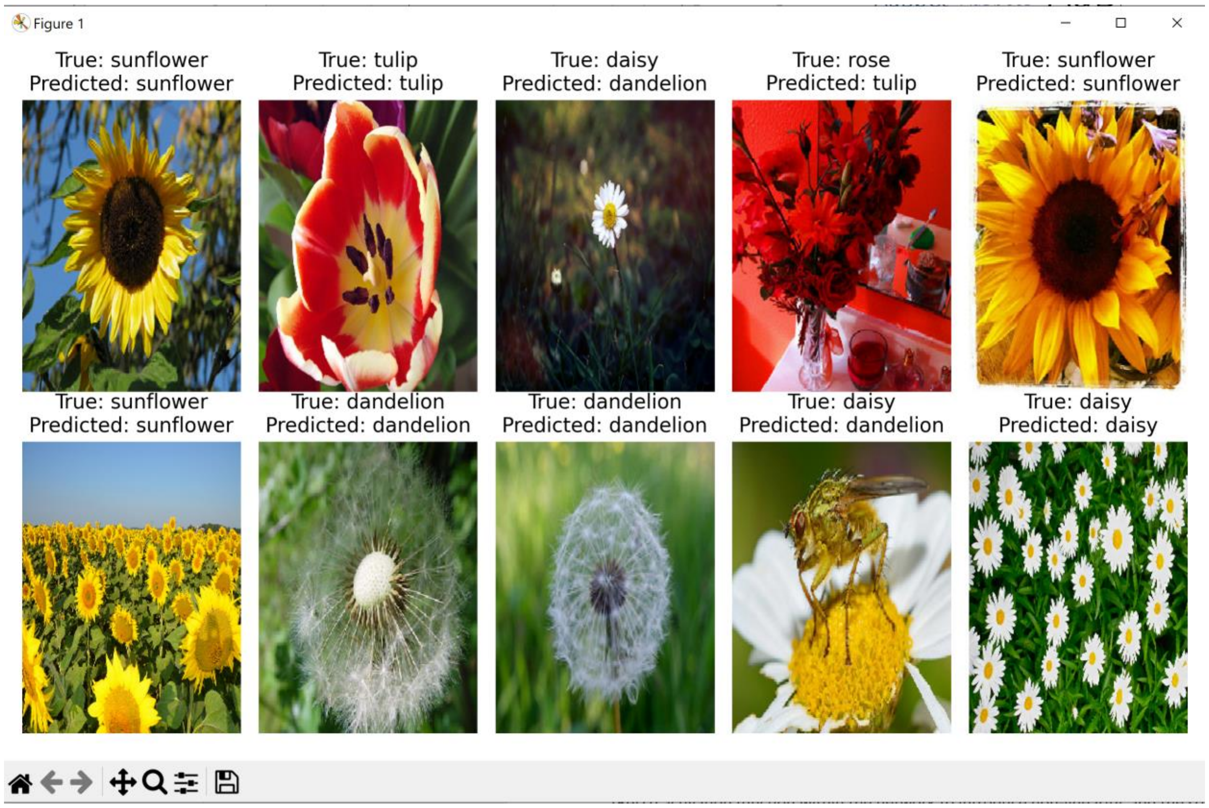
```
95/95 [==============================] - 129s 1s/step - loss: 1.3695 - accuracy: 0.4224 - val_loss: 1.1269 - val_accuracy: 0.4946
Epoch 2/10
95/95 [==============================] - 130s 1s/step - loss: 1.0768 - accuracy: 0.5650 - val_loss: 0.9961 - val_accuracy: 0.6368
Epoch 3/10
95/95 [==============================] - 130s 1s/step - loss: 0.8875 - accuracy: 0.6713 - val_loss: 0.9148 - val_accuracy: 0.6461
Epoch 4/10
95/95 [==============================] - 120s 1s/step - loss: 0.6960 - accuracy: 0.7408 - val_loss: 0.9476 - val_accuracy: 0.6584
Epoch 5/10
95/95 [==============================] - 108s 1s/step - loss: 0.5025 - accuracy: 0.8196 - val_loss: 0.9887 - val_accuracy: 0.6723
Epoch 6/10
95/95 [==============================] - 109s 1s/step - loss: 0.3293 - accuracy: 0.8851 - val_loss: 1.1097 - val_accuracy: 0.6522
Epoch 7/10
95/95 [==============================] - 114s 1s/step - loss: 0.2358 - accuracy: 0.9176 - val_loss: 1.3362 - val_accuracy: 0.6522
Epoch 8/10
95/95 [==============================] - 106s 1s/step - loss: 0.1458 - accuracy: 0.9527 - val_loss: 1.6696 - val_accuracy: 0.6260
Epoch 9/10
95/95 [==============================] - 110s 1s/step - loss: 0.1296 - accuracy: 0.9583 - val_loss: 1.7889 - val_accuracy: 0.6507
Epoch 10/10
95/95 [==============================] - 108s 1s/step - loss: 0.0999 - accuracy: 0.9669 - val_loss: 1.8897 - val_accuracy: 0.6553
21/21 [==============================] - 5s 236ms/step - loss: 1.9693 - accuracy: 0.6364
Test Loss: 1.9693169593811035
Test Accuracy: 0.6363636255264282
```

Loss and accuracy curves:





Sample Testing Output (only the first 10 images from the test set):



Effectiveness of Network

My network regularly attains between 55% and 70% testing accuracy. This is sufficient accuracy for a locally ran image classification CNN with 5 classes and a limited dataset. It performed relatively well because of the model design and parameters I used. The convolution and pooling layers effectively extract the features of the images, and the dense layers effectively consolidate these learnings into a reasonable probability distribution. The usage of the Rectified Linear Unit (ReLU) activation function within the network to introduce non-linearity, and the softmax activation function to produce a final probability distribution worked together well. The classic CNN setup of using some convolution layers for feature extraction, then some fully connected layers for classification, resulting a probability distribution was effective in achieving my goal. Overall, my network was effective in classifying floral images in the dataset.

I do not think my network would do as well with images outside the dataset. I used a single dropout layer to improve the generalization capability of my model, and this helped within the context of the dataset, but likely would not be as effective testing outside the dataset. I also stuck with only 10 epochs and 3 convolution layers to train the network, which could likely be optimized more for using images outside the dataset. Overall, I predict that my network would work using images outside my dataset, but not be as effective.

Conclusion

Conclusion Summary

The goal of this project is to implement a deep learning technique called convolutional neural networks (CNNs) for image classification. I utilized existing Tensorflow Keras APIs, along with other standard Python modules create an effective program. My approach was to load in data from the provided dataset, organize and preprocess it into usable data structures, construct a neural network using common standards and concepts taught throughout the project lessons, then train and test my network. I was able to achieve sufficient results with 55% to 70% testing accuracy. I was able to combine my learnings from class with a well-known machine learning framework to create an effective convolutional neural network.

References

[1] F. Moreno, "Building a Convolutional Neural Network (CNN) in Keras," Towards Data Science, [Online]. Available: https://towardsdatascience.com/building-a-convolutional-neural-network-cnn-in-keras-329fbbadc5f5. Accessed on: Dec. 17, 2023.

[2] P. Solai, "Sparse Categorical Cross-Entropy vs Categorical Cross-Entropy," Medium, [Online]. Available: https://fmorenovr.medium.com/sparse-categorical-cross-entropy-vs-categorical-cross-entropy-ea01d0392d28. Accessed on: Dec. 17, 2023.

[3] J. Allibhai, "Convolutions and Backpropagations," Medium, [Online]. Available: https://pavisj.medium.com/convolutions-and-backpropagations-46026a8f5d2c. Accessed on: Dec. 18, 2023.

[4] T. Kurbiel, "Derivative of the Softmax Function and the Categorical Cross-Entropy Loss," Towards Data Science, [Online]. Available: https://towardsdatascience.com/derivative-of-the-softmax-function-and-the-categorical-cross-entropy-loss-ffceefc081d1. Accessed on: Dec. 18, 2023.

[5] Sidharth, "Backward Pass in Convolutional Neural Network Explained," PyCodeMates, [Online]. Available: https://www.pycodemates.com/2023/07/backward-pass-in-convolutional-neural-network-explained.html. Accessed on: Dec. 18, 2023.

[6] H. Leung, A. Arnab "Final Project Overview" ENEL 252, University of Calgary, Calgary Dec 16, 2023.