Selected topics in Computer Science-2 Project Documentation

In our project “Multi-Class classification using CNN” which was based on the Paper “Multi-Class Image Classification using CNN and Tflite”.

The paper was published in “International Journal of Research in Engineering, Science and Management” on November 2020.

The paper was about Convolutional neural networks and how to implement multiple convolutional layers to increase accuracy and efficiency of the model with the proper dataflow using Keras and TensorFlow.

The paper used fashion_Mnist to implement the model and test it on Mnist dataset and got an accuracy of around 91% on training images and on validation images they got 89% accuracy.

So now let’s move on to the details of our implementation of the paper.

We started our project by importing the important libraries we later used in our model such as:

- Matplotlib.pyplot
- Numpy
- Tensorflow
- Keras
- Tensorflow_datasets

Then we chose our dataset very carefully to get the best accuracy out of our model, we chose the “mnist_corrupted” dataset from tensorFlow datasets website which contains 70,000 images: 10,000 train and 60,000 test images, number of output classes are 10 classes and here are all the information you need to know about the dataset:

```
print(info)
```

```
tfds.core.DatasetInfo(
    name='mnist_corrupted',
    version=1.0.0,
    description='MNISTCorrupted is a dataset generated by adding 15 corruptions to the test
images in the MNIST dataset. This dataset wraps the static, corrupted MNIST
test images uploaded by the original authors',
    homepage='https://github.com/google-research/mnist-c',
    features=FeaturesDict({
        'image': Image(shape=(28, 28, 1), dtype=tf.uint8),
        'label': ClassLabel(shape=(), dtype=tf.int64, num_classes=10),
    }),
    total_num_examples=70000,
    splits={
        'test': 10000,
        'train': 60000,
    },
    supervised_keys=('image', 'label'),
    citation="""@article{mu2019mnist,
      title={MNIST-C: A Robustness Benchmark for Computer Vision},
      author={Mu, Norman and Gilmer, Justin},
      journal={arXiv preprint arXiv:1906.02337},
      year={2019}
    }""",
    redistribution_info=,
)
```

Then we split the dataset into two parts which are train and test.

After that we started preprocessing the data, turning images into greyscale, changing the type to float32 and dividing all images by 255 to normalize all the data (get the values between 0 and 1).

Then we started building the model and implement the network which was in the paper layer by layer, so here's the dataflow for the network in the paper:

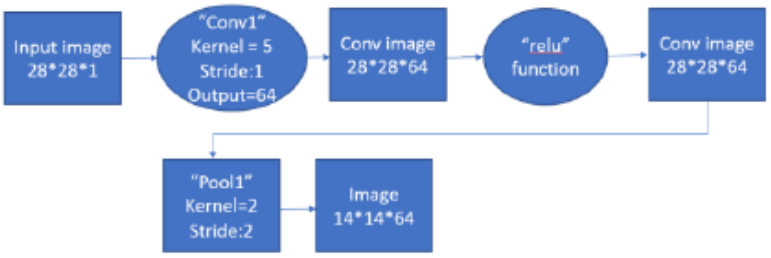

Fig. 5. First Convolution Layer

2) Dataflow for a second convolutional layer

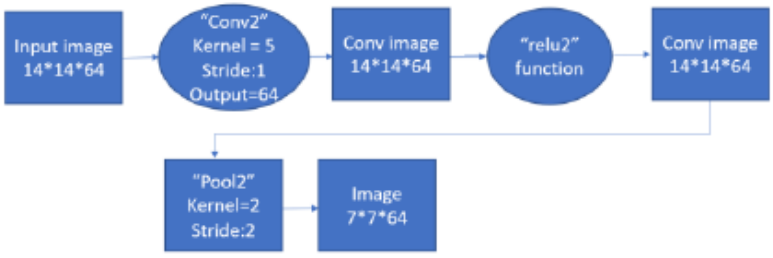

Fig. 6. Second Convolution Layer

3) Dataflow for a third convolutional layer

And here’s the network we built:

```
model = keras.Sequential([
    keras.layers.Conv2D(64, 5, strides=(1,1), input_shape = (28, 28, 1), activation = 'relu'),
    keras.layers.AveragePooling2D(2, strides =(2,2),),
    keras.layers.Conv2D(64, 5, activation = 'relu'),
    keras.layers.AveragePooling2D(2, strides =(2,2)),
    keras.layers.Conv2D(32, 2, activation = 'relu'),
    keras.layers.AveragePooling2D(),
    keras.layers.Flatten(),
    keras.layers.Dense(128, activation = 'relu'),
    keras.layers.Dense(10, activation = 'softmax')
])
```

So we used keras.sequential to stack the layers one after another, then we started building the model by adding the Conv2D layers and activation function ‘relu’ to the next layer which was AveragePooling2D layer in between every Conv2D layer and the other (to reduce the dimensions of the matrix and increase the efficiency) then we flatten the whole matrix and goes to the fully connected layer from which we go to the last output layer with ‘softmax’ activation function.

Then we compiled our model using “adam” optimizer and “sparseCategoricalCrossEntropy” loss to compute the loss between predicted and real labels and we used “accuracy” metrics.

Finally we fit our model with validation split of 0.2 and 5 epochs and batch size equals to 32.

We were able to obtain a training accuracy of 98% and loss of 0.05 out of that network.

Then we tested it and we got 98% accuracy and 0.034 loss.

```
[ ]  model.evaluate(test_images, test_labels)

     313/313 [==============================] - 2s 5ms/step - loss: 0.0341 - accuracy: 0.9892
     [0.03405771031975746, 0.9891999959945679]
```

The model accuracy and model loss graphs: