Table of contents

Introduction

Project Description

Experiments/Simulations & Results

Conclusions

Neural Network Image Classification

**Introduction**

In this article we are gonna walk through a real-world example of using neural nets. The purpose is to classify images of rock, paper and scissors using a convolutional neural net written in tensorflow and Cara's .

**Dataset**

Tensorflow offers prepared datasets that we can access via the tensorflow builder. For this project we chose the ‘rock\_paper\_scissors’ dataset. The dataset contains images with 300 height x 300 width x (0~255) color channels. It is an RGB image we can learn the data type here which is

a you ain't 8 so what that ultimately is telling me is that these are numbers each of these images are numbers from 0 to 255 and then we also have the class label which is just a integer that's

basically representing

rock-paper-scissors then finally another

the data set includes 2520 images that we can use for

training and then 372 that we'll use for

our test set and our validation and

-their hands doing rock paper and scissors and one thing you might notice about these hands is that they're a little bit some of them are a little bit funky looking like this one right here I'm

these are all artificially generated images of rock paper and scissors so they're not actual images as far as I know but they work the same when we're kind of going ahead and training our network we can kind of utilize them as we'd use a a real image

labels 0 for rock

**data prep**

if this step this is really kind of for any sort of image classification task I'd say this is a pretty similar type of processing I would probably do but we're basically converting the tensorflow

datasets format into a numpy format that's a little bit more Universal a little bit more easy to work with in my opinion .

that's what we're gonna do in this next step I thought the easiest way

to figure out how to do this was to

first look at the documentation for the data sets library within tensorflow so into tensorflow org slash data sets slash overview and in there it told me how to iterate over a data set and that's

we're going to say our train imagesare going to be equal to example and then we're going to use get the image for example in the data set and that's basically all we have to do

one thing i do want to know though is what is the shape of the overall images

**Image Processing**

2520 300x300 images with this

After researching the colors don't really matter too much

so right now we have these three colour channels RGB but in my opinion we only really need one color Channel because we're really looking at trying to find edges and whatnot so I think we have three color channels we might kind of have more data than we actually need so

we're trying to reduce the number of things that the network has to learn so ultimately I think it would be nice to have this in **grayscale** so we're gonna do everything for the dimensions for the

first two dimensions so we need to take all these same points still but for the last dimension the color Channel we'll just take the first color Channel we'll just take that red color Channel and I

processing before we're ready to pass it

reshape these so I want to just say train images equals train images dot reshape and if

you remember what it was I'm just going to comment this out just to see this

again we had 2,500 180 by 300 by 300

so we're gonna reshape this one to twenty five twenty by 300 by 300 501

and the reason we have this one here is that whenever we use convolutional neural networks in

Kerris we always have to have some sort of semblance of a color channel and so by changing this to one we don't change the number of values we have at all we

just are basically letting Kerris know hey this is just a grayscale these are just grayscale images so do that and

**RGB EXPLANTION**

**normalize**

RGB I guess they're just our channel values from 0 to 255 when we're doing image classification we like to work with float values specifically between the range of 0 to 1 so one thing we're

gonna do is we're gonna say train images we're gonna say train images equals

train images dot as type and now it's going to be a float 32 instead of the int 8 that we had before so this is just basically getting us ready to be able to convert it from a scale of 0 to 1

instead of 0 to 255 and so the last step we're gonna do is say train images dividing equal by 255

so the max value we can have is 255 because RGB values are between 0 and 255 so by doing this we're scaling every value to be between 0 & 1 and this is just a good common practice that helps

you classify it helps the basically network learn better than if you use the 0 to 255 values you could leave it 0 to 255 but it's just ultimately it's gonna probably decrease your performance a bit

so it's a common step to normalize between 0 & 1 and that's what we're

the labels are all fine from before so we don't need to do any additional processing to the label so we'll just rerun this so and now if we look at training images data type we'll see it's a flirt 32s and if we actually wanted to like look at one of the values here we'll see that all the values are

between 0 & 1 and we can also even look at the shape here of a single example 300 by 300 by one **Training neural network**

ahead and train our first neural networkfor this task and I'm gonna go through this pretty quickly

let's define a model and that will be a **sequential model** and so in that we're gonna have to pass in some layers so the first layer we'll pass in is Karis layers **dense** so as the kind of

basic approach we'll make it **a fully connected no network** so let's say that

1. the first layer of the dense network is 512 when we're defining our first layer

we're gonna want to find the input shape so in our case now it's 300 x300x1 and then we'll

also want to define the **activation** and a good one to use is **relu** you okay that's

1. our next layer is going to be another dense layer so we'll do Karis layers dense and we'll make this one a little bit smaller so how about we do **256** here and then make the activation

another **relu** you and then

1. finally let's define our output layer and so the output layer is going to be the same size as you have labels that you're trying to classify it between so in our case this will be a dense layer of three because we have rock paper and scissors

and let's also define a activation layer for our output and when we're doing classification of and just identifying one label softmax is a good choice here

then we're gonna have to setup **the loss function for this so I'll do modelcompile we'll pass in the adam optimizer** we will use for a loss function we will use Karis losses dots Perce categorical **cross entropy** and then finally we will pass in our metrics and that will just be accuracy okay so we have our loss function set up and finally we need to

**FIT**

fit our data to the model so model dot fit will do train or train images train labels and then we'll do a parks let's say equal five and batch size equals 32 on that input shape is not defined equals 300 by 300 by one okay go on

**FIRST ISSUE**

all right so we got an issue invalid incompatible shapes 32 or one versus 32 300 300

and ultimately I think our issue here is because we're passing in an input shape and this is kind of a complex you know this is an actual image and it has a complex dimensionality so instead of doing this what we're gonna do and this is usually what you're going to do if you're trying to use a fully connected layer for images we're gonna do Karis dot layer so we're going to add one more layer and this is gonna be called a latten layer and basically what this does is it transforms that 300 by 300 image into just a single column which would probably be of dimension 90,000 so

that's what we're doing with that layer so now

FIXED THE PROBLEM

Training results vs Testing results

it finished and looking at our results you see that after the last epoch we had about 90% accuracy so you might be like wow like this was so easy we've already classified these hands these

rock-paper-scissors hands like very well well I think though it's pretty telling if we try to do a model dot evaluate on the test images and the test labels this will tell us how well our model

generalizes to unseen data and so if this is also high then like we're golden

**you see here it got 50% on the test data –**

**estimating what went wrong**

so there's a big disconnect you know we were able to classify 90% of the training examples properly but only 50% of the test so it's not doing a very well job on the test data and the big

disconnect here is is that we are using this fully connected layer for these 300 by 300 images and so that's a total of a 90 thousand like single I guess pixels that we are connecting to the next layer

of our network and ultimately we don't have we're putting too much importance in those little single pixels so what's happening here is that we're basically we're overfitting to our training data and we're not really learning good patterns because as you see in our test

data we got 50 percent so ultimately this is what's going to lead us to want to use convolutional neural networks to kind of get a more sense of generalizable features within our rock-paper-scissors images

**Possible fix**

so that's what we're going to do for our next network I'm not going to go through the details of a convolutional network in great depth but basically we can think of all of our images as a grid and before we are using each pixel to feed nto our network now within our grid image we are passing over a smaller grid across the entire image and basically with these smaller grids that we're passing on the in the entire image these convolutions were performing we're

learning features that occupy a little bit more space and as we think about it like images there's so much variance in how they are so we need to like at a higher level be able to pick up on those

features so that's why we're gonna pass these smaller grids over our bigger grid to hopefully pull out general features in our images all right

**ATTEMPT NUMBER 2**and so we're

gonna again start this off by defining a sequential model and this time what we're gonna do

is our first layer is going to be caressed layers thought con-con for convolution and to-day because this is a 2d image and then we have to pass in the appropriate parameters so if we look at our Google collab we need a first pass in filters so this is basically how many different smaller grids we're gonna pass on top of our image and each of these smaller grids is going to have a

different kind of shape and try to find a different pattern I'll link to some

resources that can explain this in more depth it's basically how many times we're passing over a smaller grid on our image so we're gonna say this is 64 there's just a kind of a see first

answer our kernel size this is how big our smaller grid is so if I said three and I didn't pass in three to start off and we'll leave the rides at one two one that just means they'll move one every

time so it's gonna be a sliding window of three by three all right and how about we and we'll give that an activation of RELU you just like we have been and let's now we actually do need

to define the input shape and this time it will expect a multi dimensional shape so we can do 300 by 300 by one that's totally fine now I'm going to just pass in another convolutional layer so

INSIDE ATTEMPT

we're gonna do another 2d convolution this time we're gonna make it a little bit smaller

we'll say thirty two filters with kernel size of three activation equals real you and all right I think that's good we're. having we have two convolutional layers

and now we're gonna want to pass this to a kind of the output layer and to do that we first need o **flatten** the output of all these convolutions so we're gonna **pass another flatten layer**

**RESULTS SECOND ATTEMPT**

we actually did worse than we did before with the fully connected approach so obviously we over fit again so let's see how we can improve this convolutional approach and hopefully avoid **ATTEMPT NUBER 3**

our overfitting problem all right so now that's we're going to look at this what is the main issue that we're having and why are we overfitting and this is definitely not obvious unless you've kind of started to work with image classification a bit I would say

1. the problem kind of boils down to the fact that we don't have that many training examples you know only we have under 3000 training examples .
2. also the other real big issue that I see here is that our images are 300 by 300 by one but our kernel size here the small grid that we're passing on top of the image when we're doing the convolution is just three by three so it's still like a really small rectangle

we're passing on top of the big rectangle so what we're all Tamayo and like obviously like if we're thinking about our image we don't need high-resolution images as long as like

the base features are there we should be good so ultimately what we want to do here is reduce the size of this input image before we start performing the convolutions

**Possible solutin:** on it so what we're gonna do is a **average pool** we're gonna add an average pooling layer so Karis layers dot

our pool size is going to be 6 we can pass in if we if it's square we can just pass in 1/4 in it again but we could also pass in multiple coordinates so and then our stride our stride is how far if we have a six by six box our stride is how much we move that box each time we do an average so

I'm going to say that this is three so basically what we're doing here in thislayer is that as we pass it and we're gonna also have the pass in the input shape here so what we're basically doing

is that we have this 300 by 300 input image and what we're doing is basically

passing a six by six box on top of that image and every time we do that we're a bridging all the pixels and that's our kind of new representation of that box and instead of removing the box entirely over we're moving the box three so every two times we move the box we are any

like basically new location so you basically have this and this and this and this second time you move that box here kind of covering new squares I guess from the first time you did it so I guess the better way to say this is that each pixel is counted twice in this representation but basically we're gonna boil this down if we have six and removing on top of it it's going to boil this down from 300 - 300 - I believe if I'm not mistaken 100 by 100 as the new kind of input and you could reduce this further further if you wanted to if we did six by six here with a stride of six

think this would reduce it down to 50 by50 so that's another thing to try but I

want to count each pixel twice here and then we'll feed this now into our convolutional Network so this is basically just the image with instead of size 300 by 300 after this layer it's

100 by 100 and

**RESULTS** let's see what happens to our network when we just do that reduction in image size so still like basic classifies the test data very well let's see what happens when we evaluate

it so test images test labels want to do better than last time okay 56% I mean that's definitely better I think we can still do better here and I

**ATTEMPT 4**

another very common thing in convolutional neural nets is to do some max pooling

and this is basically for the same reason is we don't want to take too many dimensions when we're doing these things so max pool 2d is going to basically do the similar thing as an average pool but now we're gonna pass over our output with two by two grids and just take the

max pixel in every 2x2 grid and this time we're moving the 2x2 grid to every time so each pixel is only counted one but we're just taking the max pixel value from each of those so let's see

what happens if we add that

**RESULT 4**

66% accuracy on the test data that's a lot better and it's ultimately because we're reducing the size of things and basically individual pixels don't have as much weight anymore so it's smaller

input space smaller parameter size so that our model should generalize more

**ATTEMPT 5**

but there's one major thing that I'm thinking of that we can do to probably help things out and that is to **use dropout** so one of our main issues with our training examples is that we don't

have many and so what dropout will do is that after we're done with the max pool 2d

layer we're going to basically cut out 50% of the connections so as we train things 50% of the connections that this lay are going to be dropped out and ultimately the epochs will be more

effective because the data will be kind of more interesting because these things are getting dropped out randomly so each of the individual connections will kind of have to generalize more and be able to handle more variation so this is going to kind of help us simulate having

more training examples by using this dropout layer and 50% is a good it's kind of in research the common value to go to with this dropout layer so let's try this

**RESULT 5**

we're about the same 65 percent didn't seem to help too much

**ATTEMPT 6**

the last thing that I'm gonna recommend doing and this is something that you just really play

around with but we're gonna add another dense layer here so instead of going from everything to three we're going to try to learn an inter minute intermediary representation before we

actually do the output so we're hoping that maybe this layer can capture some higher-level information and boil it down to the three better than just passing all the outputs of the

convolution and the dropout and everything straight to our classification layer so make this 128

we'll use see what happens if you use lü activation

**RESULT 6**

70 percent accuracy that's pretty good on the test date on scene data I'mpretty happy with that so I think as far as on your own you could tune these values and play around with adding more values and whatnot

I was able to I think get up to like 81 percent accuracy on the test data and to do that I added

some additional things so I like played around with how many convolutional layers were used and how many filters so that would look something like thi