



Image Classification Web Application

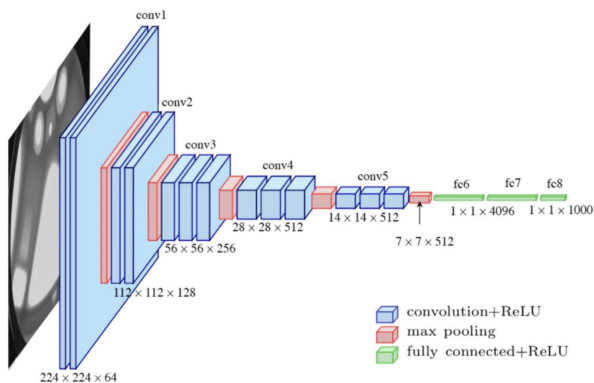
By: Yee Hang

Objective : User should learn about Convolutional
Neural Networks by interacting with the UI

Model Info

Modified VGG:

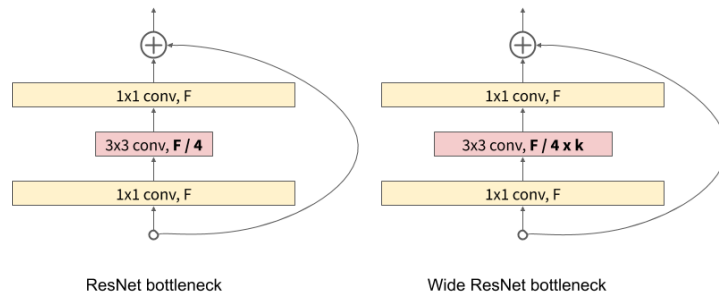
- A scaled down version of VGG with only 8 Convolutional layers with max pooling and dropout in between them, so that inference is faster, especially when hosted in cloud.



VGG

Wide ResNet:

- Contains skip connection in Wide ResNet blocks with larger architecture
- Contains 28 convolution layers, and 10 times number of filters as default ResNet
- Generally more accurate but slower inference times



Wide ResNet

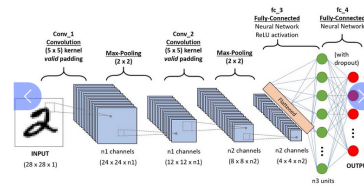
Application – Front end - Homepage

- Give the users some background into image classification with convolutional neural networks
- This page explains how the networks work and how they produce a classification

Image Classification

[Home](#) [Log In](#) [Sign up](#)

What is this Image?



[Convolutional Neural Networks](#)

[Models Used](#)

Convolutional neural networks (CNN) are a significant improvement on the Multi layer perceptron architecture used to classify images. The convolutions performed does not require the system to test all the possible combinations between individual pixels, which speeds up training.

There are many proposed (CNN) architectures which are Le Net (1998) for classifying hand written digits with a simple architecture, Alex net which introduced Max Pooling, ReLU Activation and Dropout Visual Geometry Group decreased the filter size to 3x3 filters.

Skip connections bring about a large improvement in performance, as it alleviates the vanishing gradient problem by providing better information flow. This allowed researchers to try out larger networks.

[Convolutional Neural Networks](#)

[Models Used](#)

Visual Geometry Group

The VGG used has half the number of convolutional layers (8) compared to vgg16, to reduce inference time. The implementation of VGG serves as a decent model that returns results in a shorter amount of time.

Wide ResNet

Wide ResNets are introduced by increasing width (number of filters per layer) instead of the depth of network. This leads to faster training, with smaller architecture. The implementation is chosen is the Wide ResNet 28 Convolutional layers and 10 times the number of filters as original ResNet

Cut Mix Augmentation

Cut mix Augmentation is also used in training to train both models, as it is found to be effective in preventing overfitting.

Please give the models a try by uploading pictures to the predict page

Application – Prediction Page

- Allow user to upload a photo and display the photo to the user as html canvas
- Allow user to corrupt the picture by adding black lines to obstruct the picture and compare the results of predictions with and without augmentation

Upload an image to get predictions on classifier



Upload a photo here to predict what it is

Browse...

cloud.jpeg

Predict with Wide ResNet

Predict with vgg

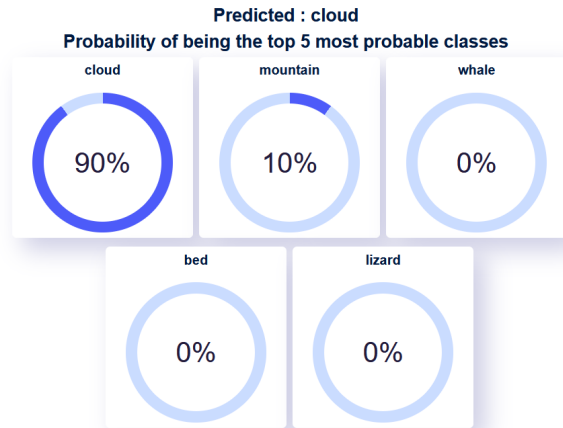


Upload a photo here to predict what it is

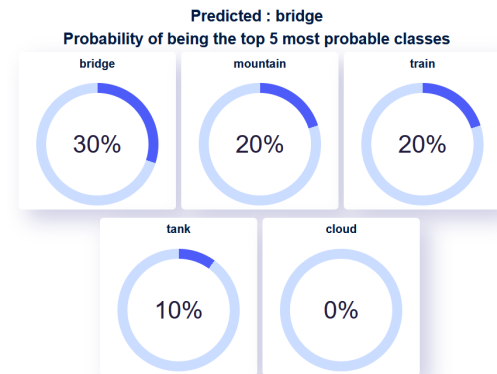
Augmenting Pictures

Application – Prediction Results Page

- The prediction results will appear below the page, once result returned from server
- Prediction probabilities of the top 5 most probable classes are also returned , so the user know how “confident” the model is about the classification
- Prediction of an image with corruption makes the model less confident about its predictions
- Non corrupted data is 90% confident and gives correct prediction, the corrupted data gives 30% and the wrong class (bridge)



Prediction without augmentation



Prediction with augmentation

Application – Prediction Image downsampling

- In the front end, Images with width larger than 1500 pixels or height larger than 1000 pixels, is rescaled to 1500 pixel width or 1000 pixel height, (To decrease image size and speed up post request to server)
- Algorithm retains aspect ratio and prevent distorting
- Compress the image with image quality argument of 'canvas.toDataURL' with 0.6
- Image is not resized directly to 32 x 32 as the resizing algorithm depends on the browser and might lead to inconsistent predictions on different browsers

```
if (imgwidth > 1500) {  
    let scalefactor = imgwidth / 1500  
    imgwidth = imgwidth / scalefactor  
    imgheight = imgheight / scalefactor  
} else if (imgheight > 1000){  
    let scalefactor = imgheight / 1000  
    imgwidth = imgwidth / scalefactor  
    imgheight = imgheight / scalefactor  
}  
  
canvas.width = imgwidth;  
canvas.height = imgheight;  
  
ctx.drawImage(img, 0, 0, imgwidth, imgheight);
```

Decrease the dimension of image in Javascript

```
var img = canvas.toDataURL('image/png', 0.6);  
console.log(img)
```

Compress the image with image quality argument of

Application – Prediction Image downsampling

- The image are downsampled to 32x32 size using the resize method of PIL.Image.
- It is then sharpened using the ImageEnhance.Sharpness function
- Image is HTTP POSTED to the tensorflow serving for classification

```
unsharped3232 = Image.open(filepath).resize((32,32))  
sharpend3232 = ImageEnhance.Sharpness(unsharped3232).enhance(2)
```

Image is read in and sharpened, using
Pil.ImageEnhance.Sharpness

Application – Prediction History Page

- Provide the image, model used, the predicted class and the date and time the prediction is done on
- Pagination added for easier navigation, prevent excessive scrolling
- Option to filter by start date ,end date and by model






Your Prediction history

Start Date
mm / dd / yyyy

End Date
mm / dd / yyyy

Model
both

Search and Filter

Image	Model	Prediction	Datetime	Delete
	vgg	whale	14 Feb 23 01:46	Remove
	wideresnet	cloud	14 Feb 23 01:50	Remove
	vgg	forest	14 Feb 23 02:38	Remove
	wideresnet	bicycle	14 Feb 23 02:38	Remove
	wideresnet	train	14 Feb 23 02:46	Remove

1

2

3

Application – Testing

- Expected fail test, test inserting invalid input
- Consistency testing, whether model return same result every time same images given
- Validity testing – test validity of data
- Range testing, different ranges of inputs are given

Application – Validity Testing

- To test the validity of the inputs and ensure only string 'vgg' and 'wideresnet' inserted into the model column of database
- Ensure no values less than 0 or larger than 99 inserted for predicted class as there are 100 classes in dataset.

```
self.model = data['model']
if data['model'] not in {'wideresnet', 'vgg'}:
    raise Exception('Model must only be wideresnet or vgg')

if data['predictedClass'] < 0 or data['predictedClass'] > 99 :
    raise Exception('Prediction must only be between 0 and 99 inclusive')
```

Check the range of the data before
data inserted into database

```
(
    dict(
        filename = 'photo2.jpg',
        predictedClass = 9,
        model = 'wideresnet'
    ),
)
```

```
(
    dict(
        filename = 'photo1.jpg',
        predictedClass = 6,
        model = 'vgg'
    )
)
```

Expected Pass Cases, all values are valid

Application – Expected Failure Testing – invalid data

```
@pytest.mark.xfail(strict=True)
```

- Expected fail cases
- Where the model name is invalid or the classes inserted are invalid

```
(
    dict(
        filename = 'photo1.jpg',
        predictedClass = 6,
        model = 'invalidmodel' #invalid model
    ),
    (
        dict(
            filename = 'photo1.jpg',
            predictedClass = -10, #invalid prediction , value is too low
            model = 'wideresnet'
        ),
        (
            dict(
                filename = 'photo1.jpg',
                predictedClass = 101, #invalid prediction, value is too high
                model = 'wideresnet'
            )
        )
    )
)
```

Application – validity testing- insert data and retrieving the data

- Insert a prediction into database and retrieving the results back (expected pass)
- Not inserting anything in database and attempt to retrieve results back (expected fail)

```
def test_get_prediction(application_client, data, create_pred=True):
    context = application_client["context"]
    userid = application_client["userid"]
    newdata = data.copy()

    newdata["userid"] = userid
    if create_pred == True:
        response = context.post(
            "/api/storeprediction",
            data=json.dumps(newdata),
            content_type="application/json",
        )

        newid = json.loads(response.get_data(as_text=True))["id"]
    else:
        newid = 0
    response = context.get(f"/api/getpred/{newid}")
    assert response.status_code == 200
    respondedata = json.loads(response.get_data(as_text=True))
    assert respondedata["data"]
    respondedata = respondedata["data"]
    print(respondedata)
    assert respondedata["filename"] == data["filename"]
    assert respondedata["predictedClass"] == data["predictedClass"]
    assert respondedata["model"] == data["model"]
```

When create_pred is True, data is inserted,
When create_pred is False, data is not
inserted before retrieving

Application – validity testing – inserting data and deleting the data

```
@pytest.mark.xfail(strict=True)
```

- Insert a prediction into database and deleting the row data (expected pass)
- Not inserting anything in database and attempt to delete the data (expected fail)

```
def test_delete_prediction(application_client, data, create_pred=True):
    context = application_client["context"]
    userid = application_client["userid"]
    newdata = data.copy()

    newdata["userid"] = userid
    if create_pred == True:
        response = context.post(
            "/api/storeprediction",
            data=json.dumps(newdata),
            content_type="application/json",
        )

        newid = json.loads(response.get_data(as_text=True))["id"]
    else:
        newid = 0
    response = context.delete(f"/api/remove/{newid}")
    assert response.status_code == 200
```

When create_pred is True, data is inserted,
When create_pred is False, data is not
inserted before retrieving

Application – User Validation - Expected fail test

```
regex = re.compile(r'([A-Za-z0-9]+[._-])*[A-Za-z0-9]+@[A-Za-z0-9-]+(\.[A-Z|a-z]{2,})+')  
if not re.fullmatch(regex, self.username):  
    raise Exception('Username must be in email format')  
  
if len(data['password']) < 6:  
    raise Exception('Password must be 6 characters long')
```

Application checks for the correct username and password format

```
@pytest.mark.xfail(strict=True) # Expected Fail email  
@pytest.mark.parametrize(  
    "data",  
    [(dict(username="invalidemail", password="aas")), #both email and password invalid  
     (dict(username="", password="")), # both email and password empty  
     (dict(username="valid@gmail.com", password="smal")), # valid email but invalid password  
     (dict(username="lfkdjal;fjdlas;jl", password="validlongpassword"))], # invalid email but valid password  
)
```

- test inserting into database invalid email format when users sign up for new account
- Either invalid email or invalid password should be rejected

Application – Consistency Testing

```
response = context.post('/api/getprediction', data = json.dumps( data) ,content_type="application/json")
initial_pred = json.loads(response.get_data(as_text= True))
for _ in range(50):
    response = context.post('/api/getprediction', data = json.dumps( data) ,content_type="application/json")

    newpred = json.loads(response.get_data(as_text= True))

    assert newpred['predicted'] == initial_pred['predicted']
```

- Important for model to output the same predicted class, every time the same inputs are given
- Test if the model is consistent on different image types (png , jpg and webm) and for both models

```
[
    (
        dict(
            filepath = 'test_img_1.jpg',
            model = 'wideresnet'
        )
    ),
    (
        dict(
            filepath = 'test_img_1.jpg',
            model = 'vgg'
        )
    ),
    (
        dict(
            filepath = 'lion.webp',
            model = 'wideresnet'
        )
    ),
    (
        dict(
            filepath = 'lion.webp',
            model = 'vgg'
        )
    ),
    (
        dict(
            filepath = 'fox.png',
            model = 'wideresnet'
        )
    ),
    (
        dict(
            filepath = 'fox.png',
            model = 'vgg'
        )
    )
]
```

Application – Deployment - CI/CD

- Feature branches are merged with main quickly.
- Get all the versions of all package/module/library by doing `pip freeze > requirements.txt`
- All pytest unit tests are run before each deployment '`pytest -v`', if there are failing cases, don't push and fix the bugs first.
- To Prepare the docker file, used for build the image by installing all dependencies in `requirement.txt`, exposing the port 5000 and the gunicorn server listens on that port.
- Committing and pushing to the main branch will cause render to automatically deploy.



```
RUN pip install -r requirements.txt
ADD . /app
EXPOSE 5000
CMD gunicorn --bind 0.0.0.0:5000 app:app
```




Thank You