Technische Universität Braunschweig

# Deep Learning for Remote Sensing

## Floodnet Classification

Arjun Arora
Data Science

# Outline

- Introduction
- Motivation & Objective
- Post disaster visual scene understanding
- Floodnet dataset & Classification
- Data preparation & preprocessing
- Model training
- Transfer learning
- Experimental setup & Parameters
- Results and predictions
- Conclusion
- Future works and tasks
- References

# Introduction

How can computer vision help in Disaster Management?

Rapid floods more common due to climate change


Fig 1: Pakistan floods drone images

# Introduction

How can computer vision help in Disaster Management?

Accurate, timely and understandable information - quick response and recovery on large scale.

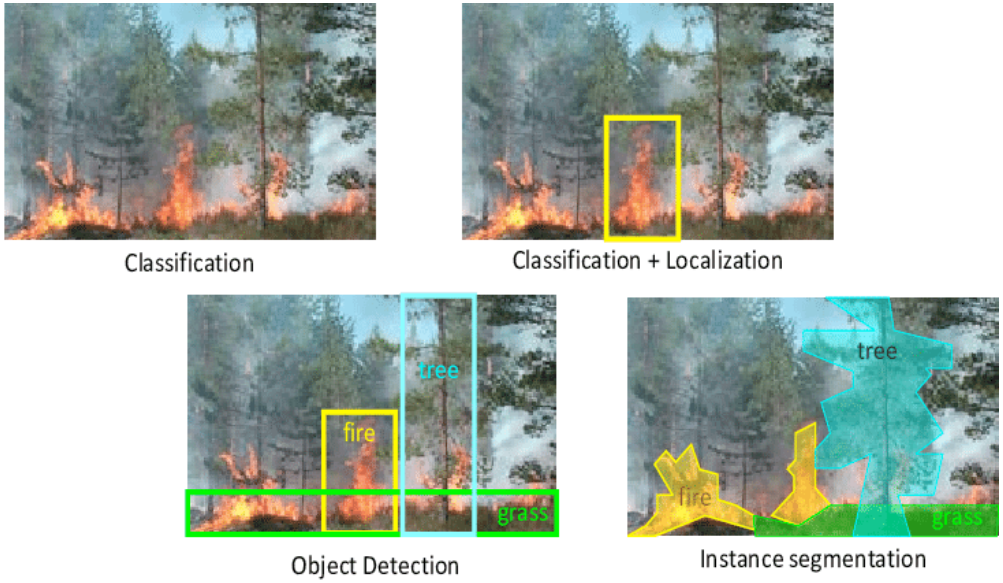



Fig 2: Computer vision in action

# Introduction

- Computer vision can help in visual scene understanding post disaster.
- Unmanned Aerial Vehicle(UAV) - access, cheap, HQ images.
- Tasks: Classification, Image Segmentation, Visual Question Answering.
- Deep learning algorithms can analyse the impact of any disaster.
- Understanding of the affected areas and plan the response.

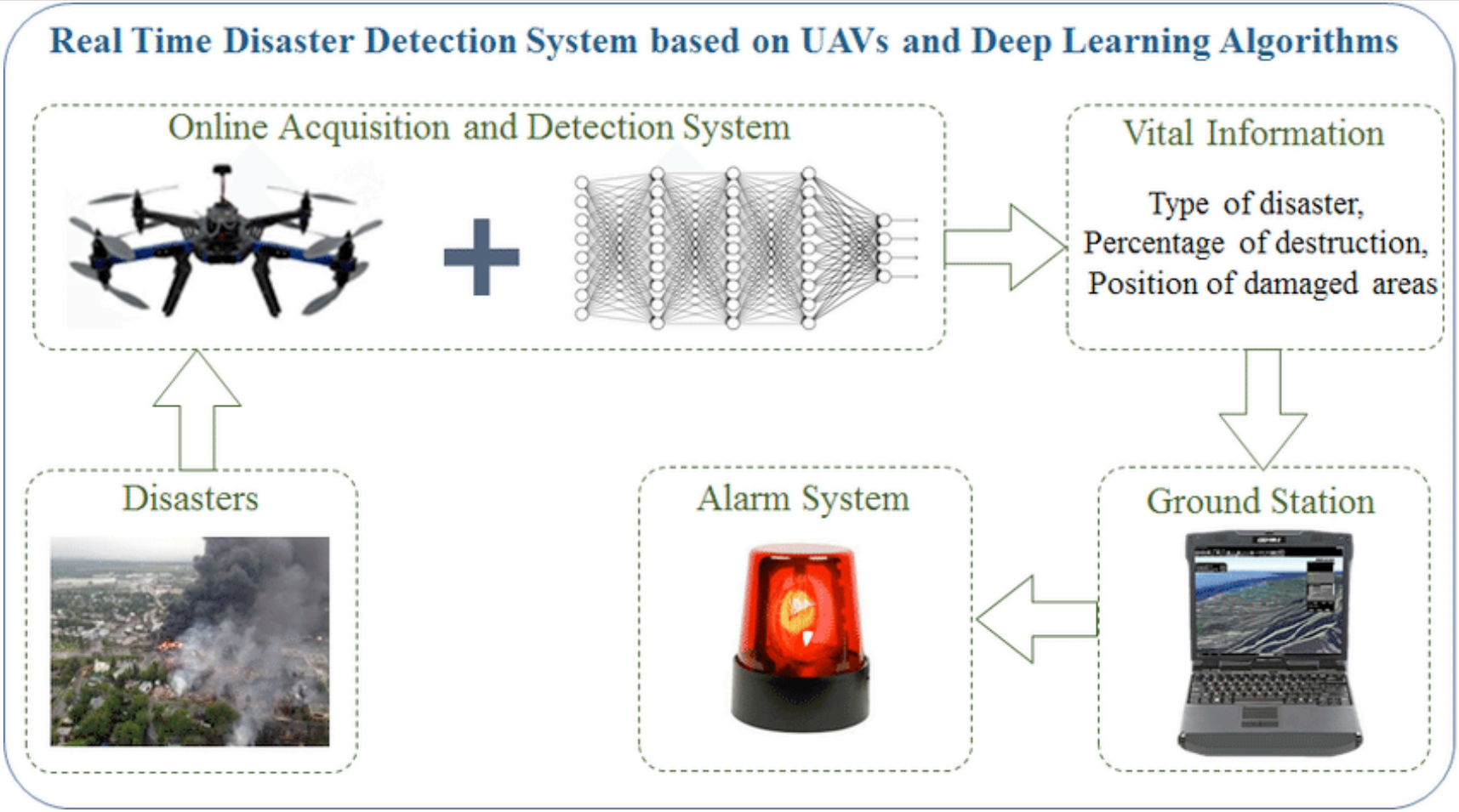



Fig 3: Disaster detection by deep learning

# Motivation

- Recent decades, floods are much more common.
- Hurricane Katrina (Florida and Louisiana, August 2005), Hurricane Leslie (France, October 2018), the 2018 monsoon season in India (Kerala, August 2018) and the floods in **Pakistan right now**

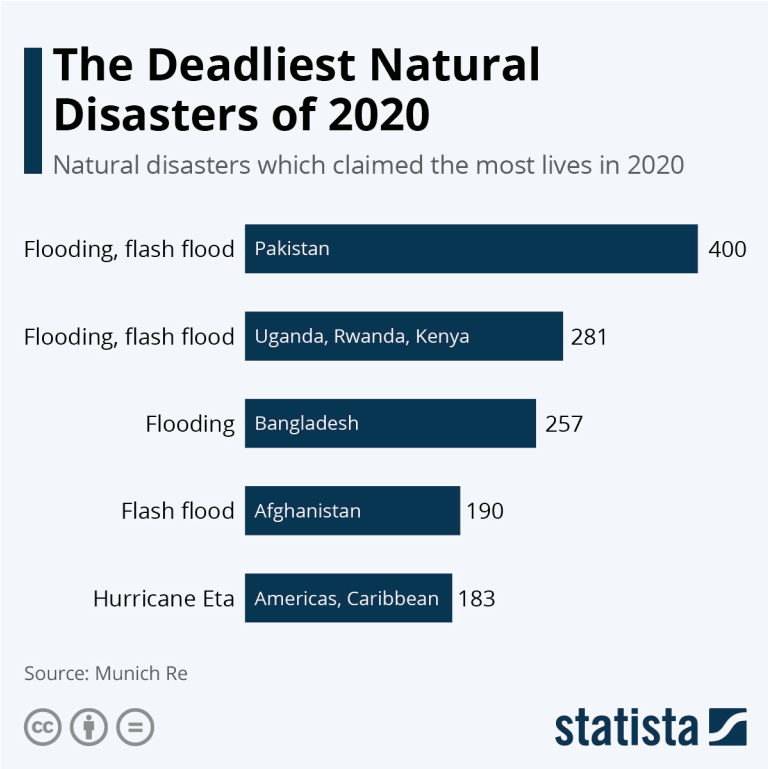



Fig 4: Deadliest natural disasters

# Motivation

- Drone is a great tool in mapping, search and rescue, transportation and training.
- Unmanned, easy, safe, HQ images.
- Provides data for quick response in the affected areas.
- NEED: Deep learning models to automatically flag high priority areas post disaster

Fig 5: Flood mapping using drones

# Post disaster visual scene understanding

- Classification - Flooded vs Non-Flooded
- Semantic segmentation - flooded roads, buildings, natural water
- Visual question answering -
  - What is the overall condition of the given image?
  - How many buildings are flooded?
  - Condition of the road in this image?

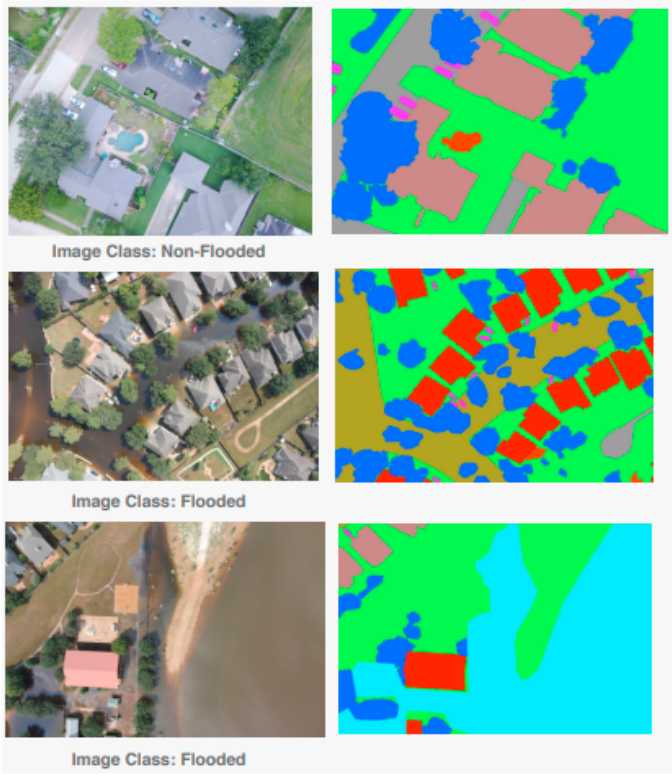



Fig 6: Some examples from Floodnet

# Post disaster visual scene understanding- Classification

•Supervised classification - Imagenet, ResNet, EfficientNet
•ResNet - Residual connection architecture; Skip connections solve vanishing gradient.
•EfficientNet - Simple but scalable

| Model | Training Accuracy | Test Accuracy | #params |
|---|---|---|---|
| InceptionNetv3 | 99.03% | 84.38% | 23.8M |
| ResNet50 | 97.37% | 93.69% | 25.6M |
| Xception | **99.84%** | 90.62% | 22.9M |
| **ResNet18 (our)** | 96.69% | **96.70%** | **11.6M** |

Table 1. Classification models comparison

Table 1: Authors classification results

# Post disaster visual scene understanding- Semantic segmentation

- Supervised Semantic Segmentation: Extract max info from image.
- Classify each pixel to a class label.
- Models - UNet, PSPNet and DeepLab.

Table 2. Number of images and instances corresponding to different classes.

| Object Class | Images | Instances |
|---|---|---|
| Building-flooded | 275 | 3573 |
| Building-non-flooded | 1272 | 5373 |
| Road-flooded | 335 | 649 |
| Road-non-flooded | 1725 | 3135 |
| Vehicle | 1105 | 6058 |
| Pool | 676 | 1421 |
| Tree | 2507 | 25889 |
| Water | 1262 | 1784 |

Table 2: Segmentation classes

# Post disaster visual scene understanding- VQA

•VQA framework: Generate questions related to building, road, and entire image.

•Precise damages and situations.

•Find right questions and multiple for each image.

•Challenge: Annotation is difficult

# Floodnet Dataset

- Source - small DJI UAV drone after Hurricane Harvey, 2017.
- Images taken during response - realistic and state of practice.
- High resolution images ~ 4/5 Megabytes.
- 1450 train images (2343 total)
- 400 images are labeled.
- Our sample set: Train labeled images
(51 Flooded, 357 Non-flooded)

# Floodnet Dataset - Challenges

- Labeled data samples are small.
- Images are very high resolution, difficult to process.
- Resize without loss of efficiency.
- Natural water bodies

Fig 7: Data sample

# Floodnet Classification

Classifying hurricane Harvey images as Flooded or Non-Flooded

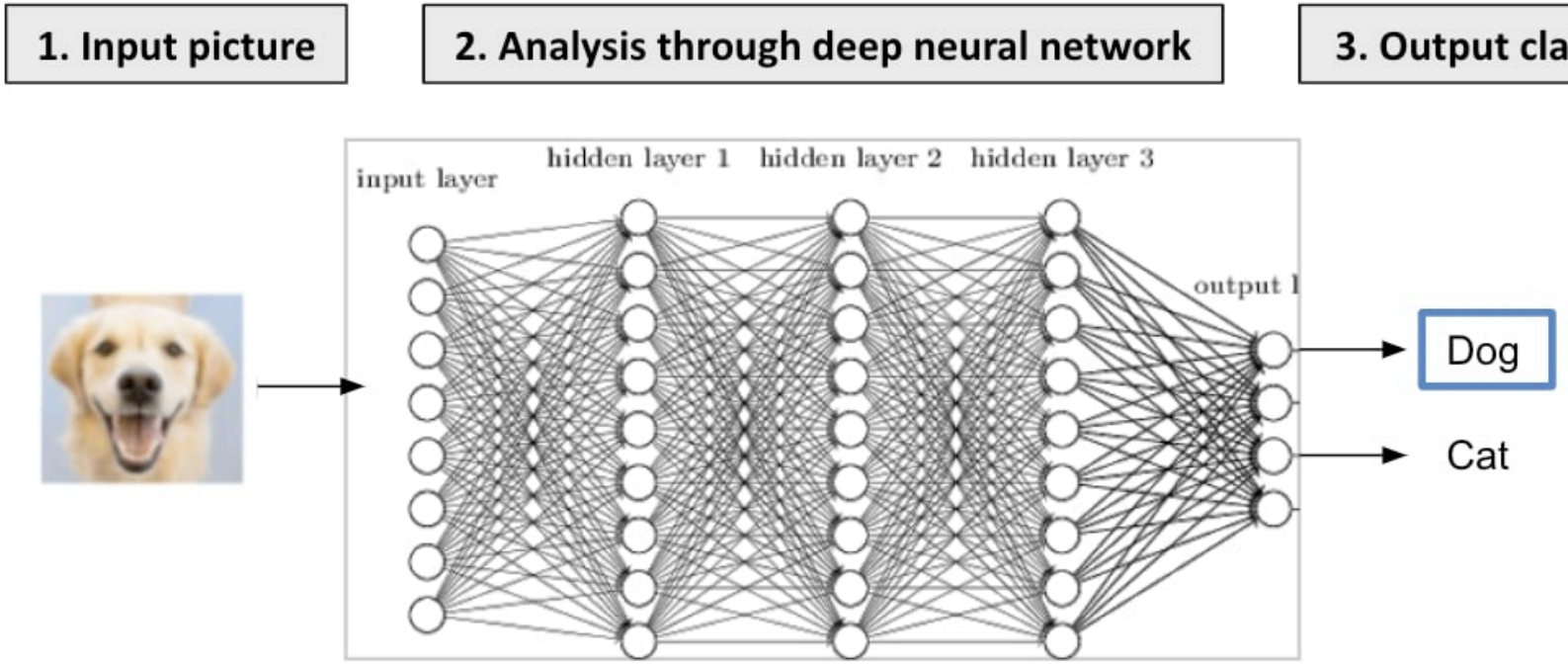



Fig 8: Image classifier

# Floodnet Classification

Classifying hurricane Harvey images as Flooded or Non-Flooded

•Steps taken for efficient and accurate classifications:

- Resize the images.
- Split dataset to train, test and validation sets.
- Use ImageDataGenerator - Augmentation and batches.
- Use transfer learning(Resnet 50)
- Test on batches of images.
- Use tensor-board for visualisation and experimentation.

# Data preparation & preprocessing

•**Resize the images:** OG image size ~ 7 MBs. Difficult to work with. Large number of parameters working with 4000*3000 pixel images.

•Ram over crashes.

•Solution! Reshape to 400*300

```
# Resize the train flooded and non flooded folders from (4000,3000) to (400,300)
RESIZE=(400,300)
temp_root = "Labeled"
local_root = "output_reshaped"

def resize_and_save(path, resize=RESIZE):
  for img_name in tqdm(os.listdir(os.path.join(temp_root, path))):
    img = cv2.imread(os.path.join(temp_root, path, img_name))
    img = cv2.resize(img, RESIZE)
    cv2.imwrite(os.path.join(local_root, path, img_name), img)
```

Fig 9: Resize the images

# Model - ResNet

•Deep residual networks(ResNet-50): 50 layer deep CNN.

•Stacks residual blocks on top of each other to form a network.

•Pretrained on ImageNet database.

•Output layer with sigmoid activation function.

```python
from tensorflow.keras.applications import ResNet50

# Initialize the Pretrained Model
feature_extractor = ResNet50(weights='imagenet',
                             input_shape=(400, 300, 3),
                             include_top=False)

# Set this parameter to make sure it's not being trained
feature_extractor.trainable = False

# Set the input layer
input_ = tf.keras.Input(shape=(400, 300, 3))

# Set the feature extractor layer
x = feature_extractor(input_, training=False)

# Set the pooling layer
x = tf.keras.layers.GlobalAveragePooling2D()(x)

# Set the final layer with sigmoid activation function
output_ = tf.keras.layers.Dense(1, activation='sigmoid')(x)

# Create the new model object
model = tf.keras.Model(input_, output_)

# Compile it
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])

# Print The Summary of The Model
model.summary()
```

Fig 10: Model preparation

# Model - Components

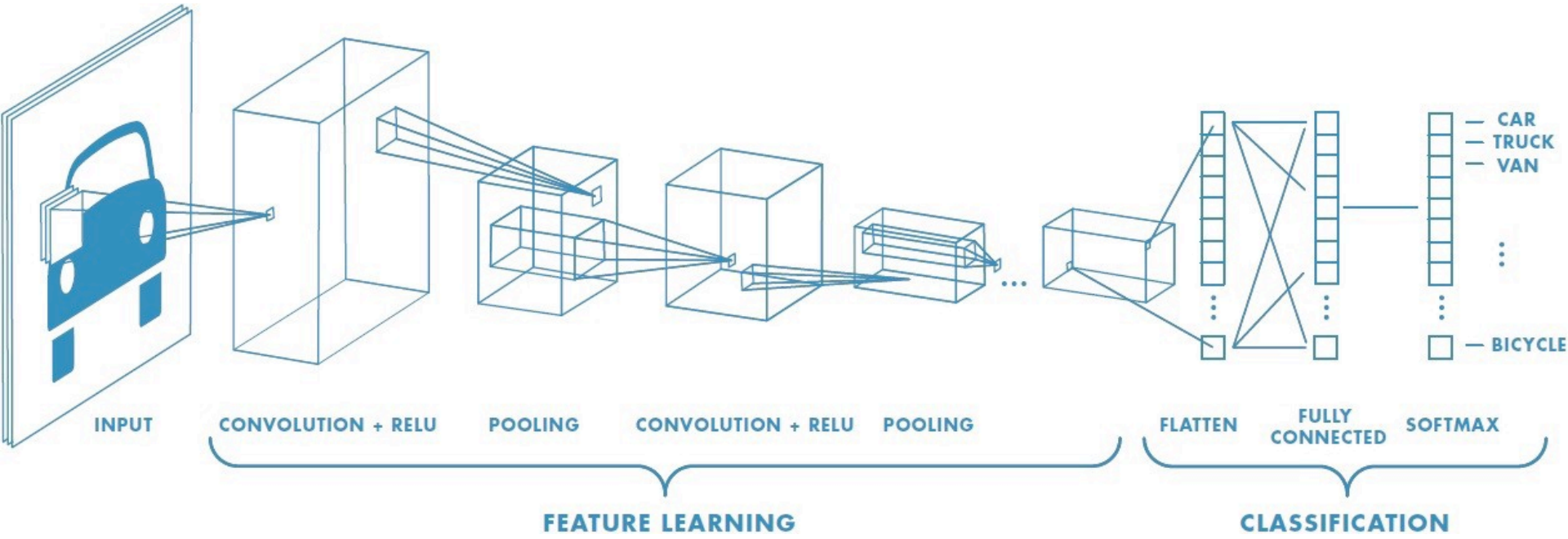



Fig 11: Image classification with conv nets

# Model - Components

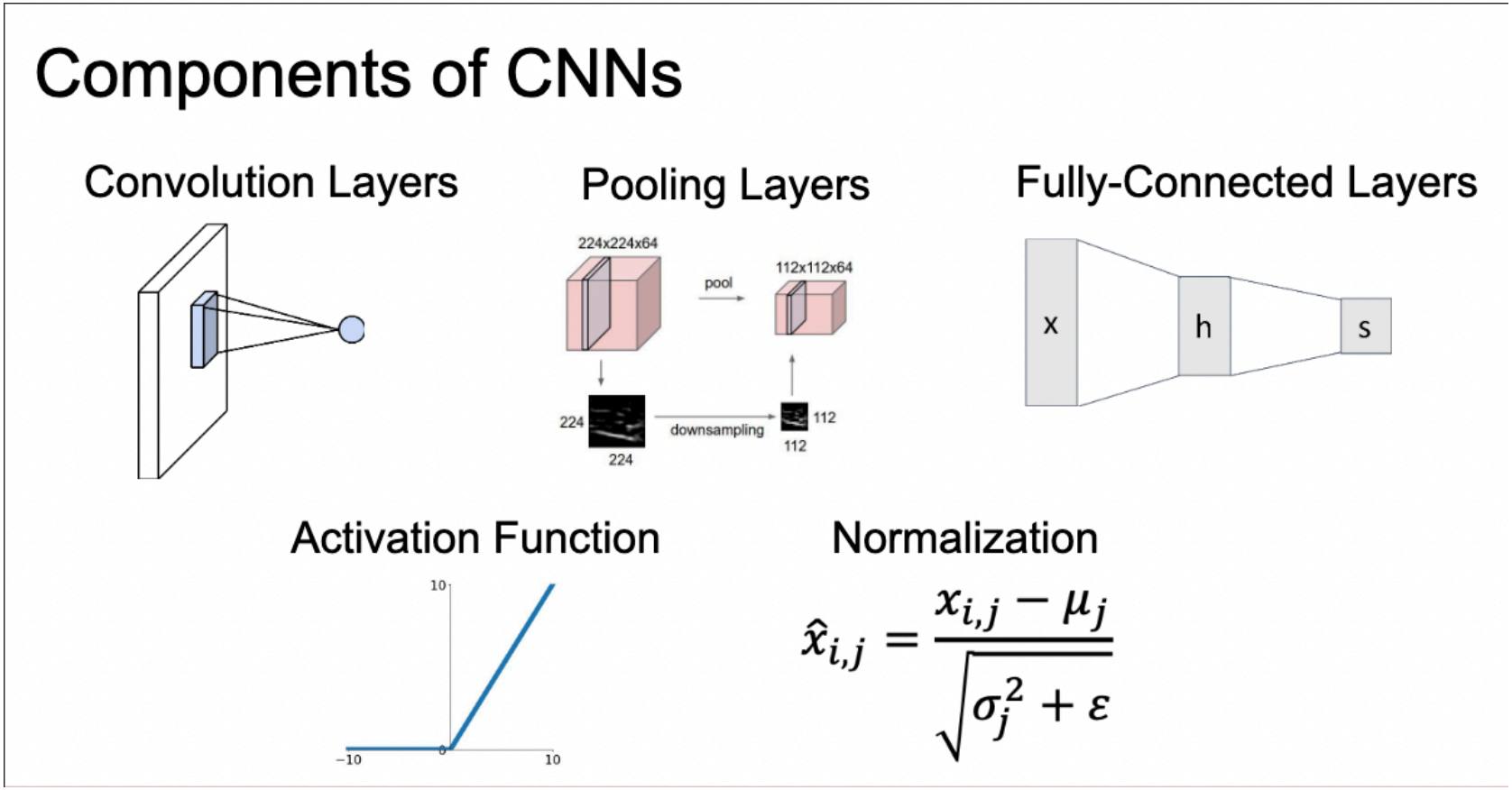



Fig 12: Components of CNN

# Model - Architecture

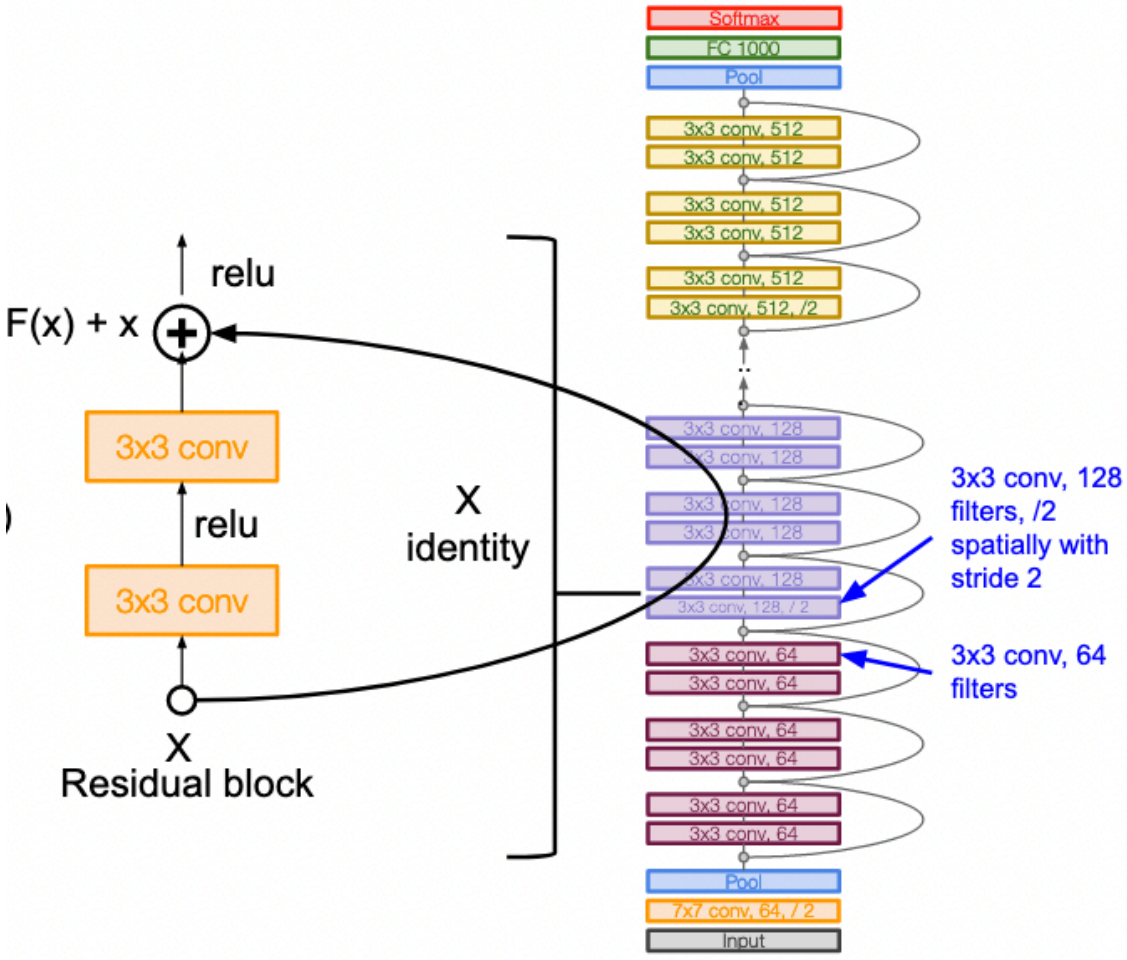



Fig 13: Resnet Architecture

# Model - Imagenet

•ImageNet - 14 million images into 1000 object categories.

•Used  in visual object detection since 2010.

•A lot of major CNNs have used this dataset to train.

•*These networks also demonstrate a strong ability to generalise to images outside the ImageNet dataset via transfer learning, such as feature extraction and fine-tuning.*

Fig 14 :Imagenet examples

# Transfer Learning

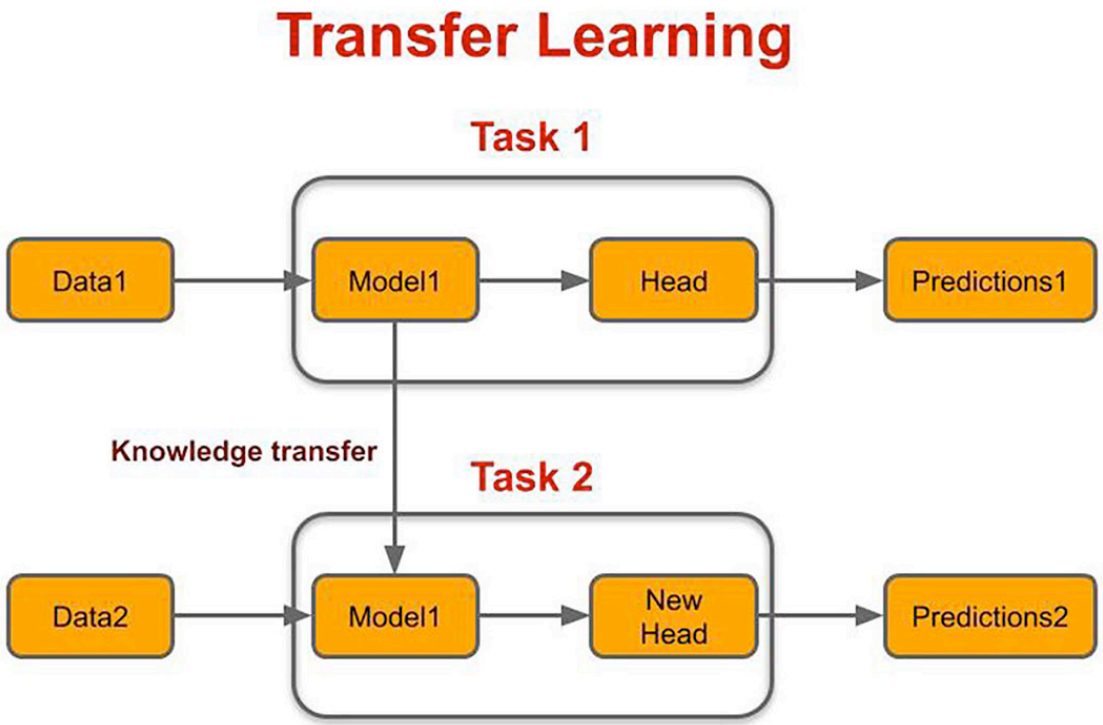



Fig 15: Transfer learning

# Transfer Learning

- Reuse pre-trained model on a new problem.
- Little data, quick train times.
- Used in Computer vision and NLP.
- You need to add/modify some layers according to the new task.

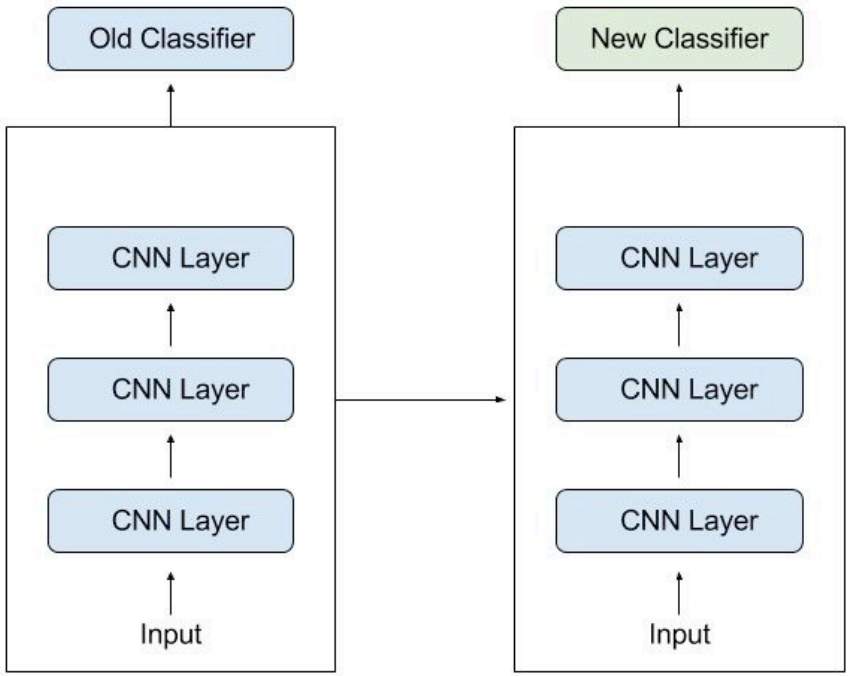



Fig 16: Learning transfer from old task to new task

# Experimental setup

•Train - 80%, test - 10%, val - 10%.

•Labels 1 for Flooded, 0 for Non-flooded.

•Input layer + Resnet50 feature extraction layer + Global average pooling layer + Output layer with sigmoid activation.

•**Integrated Feature Extractor**: The pre-trained model, or some portion of the model, is integrated into a new model, but layers of the pre-trained model are frozen during training.

# Experimental setup - Model Summary

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_4 (InputLayer)         [(None, 400, 300, 3)]     0

resnet50 (Functional)        (None, 13, 10, 2048)      23587712

global_average_pooling2d_1   (None, 2048)              0
(GlobalAveragePooling2D)

dense_1 (Dense)              (None, 1)                 2049

=================================================================
Total params: 23,589,761
Trainable params: 2,049
Non-trainable params: 23,587,712
_________________________________________________________________
```

Fig 17: Model summary

# Experimental setup - Layers

•**GlobalAveragePooling layer** - Replace fully connected layer. Take average of each feature map and the result is fed directly to activation.

•**Sigmoid activation function** - Only two classes so like 2-element softmax. Output always between 0 and 1.

•Adam optimiser, binary cross_entropy and metrics.

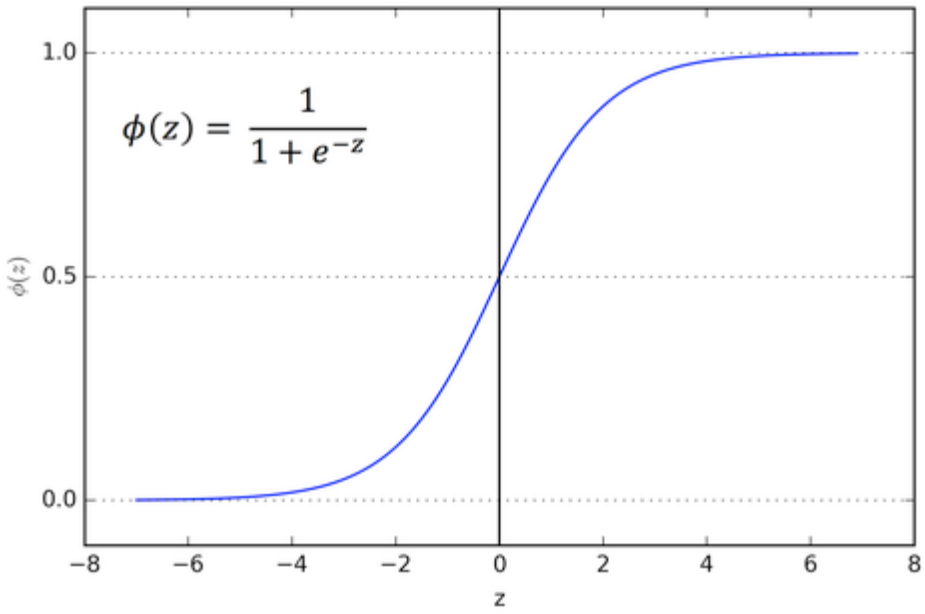



Fig 18: Sigmoid function

# Experimental setup - Model compile

•**Optimiser** - Adam optimisation is a stochastic gradient descent method that is based on adaptive estimation of first-order and second-order moments.

•**Loss function** - Binary crossentropy, binary classification tasks.

•**Metrics** - Acuuracy. Pred vs actual match. Total and count. Binary accuracy.

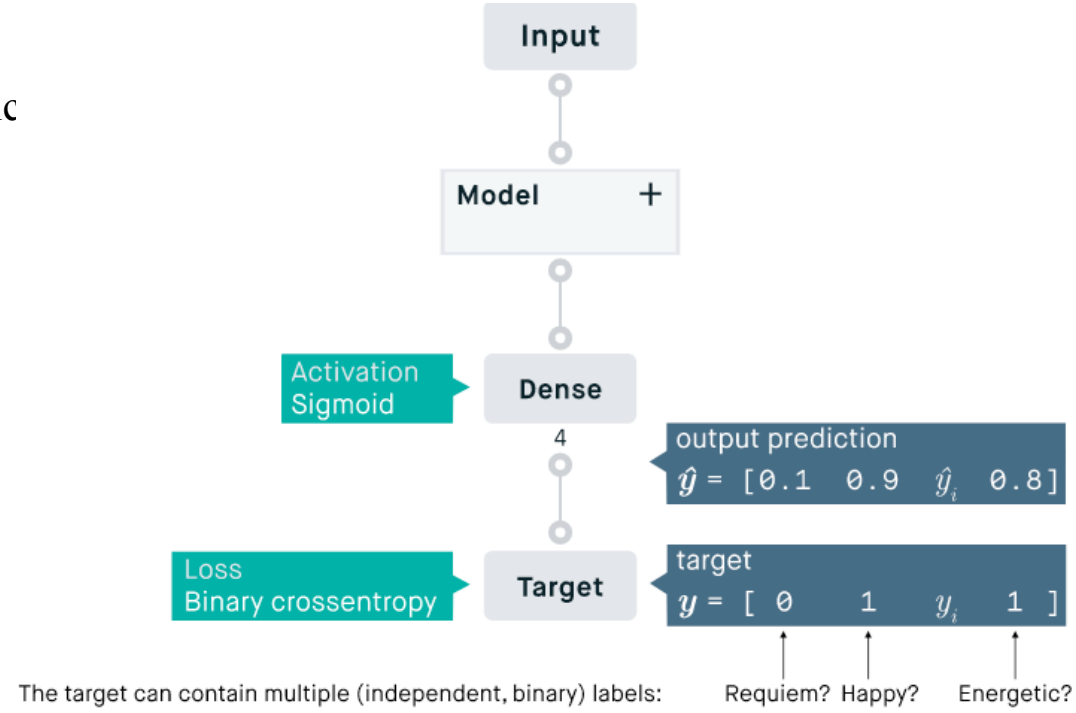
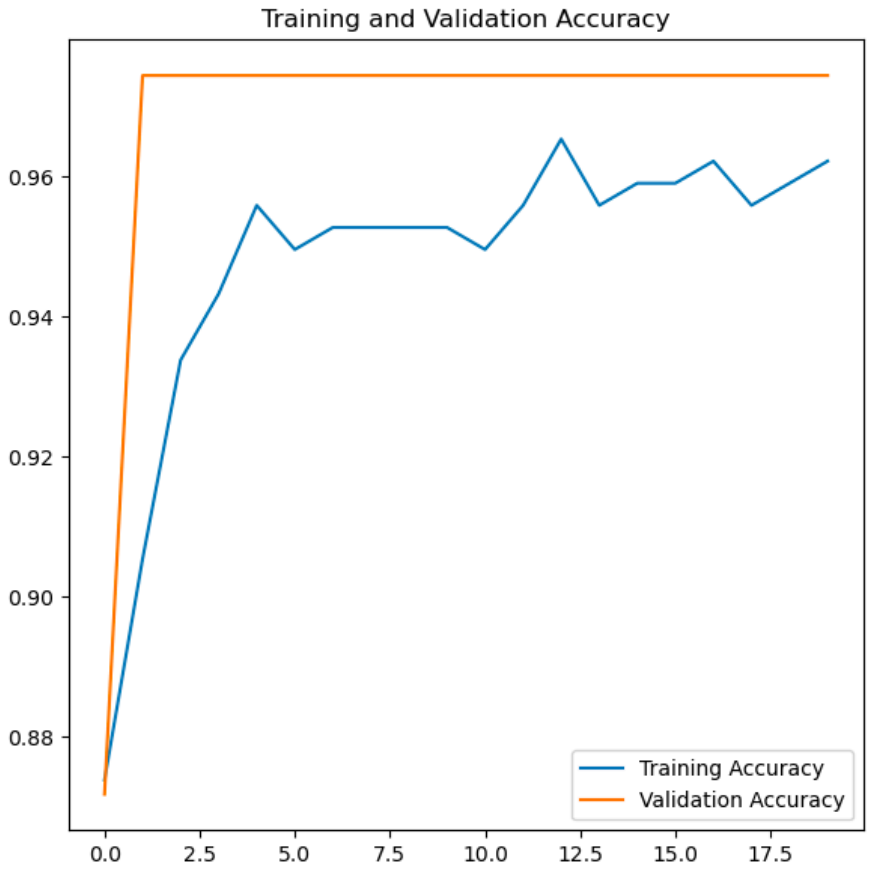



Fig 19: Binary crossentropy

# Results - Train & Val



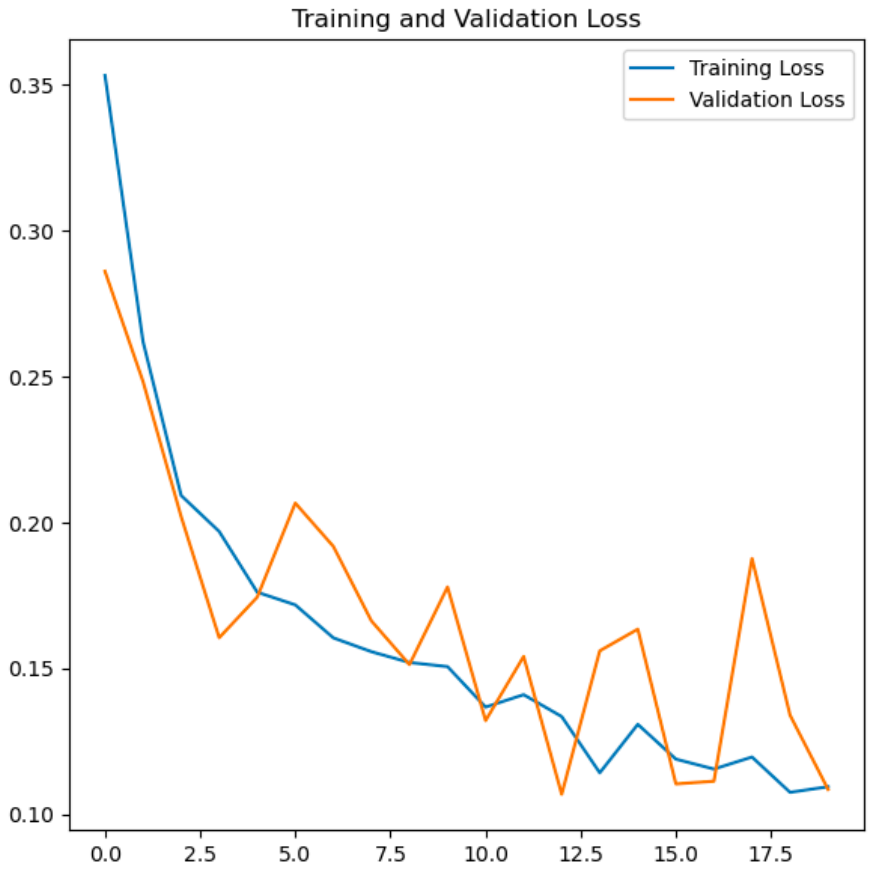




Fig 20: Accuracy and Loss trends

# Results - Tensorboard

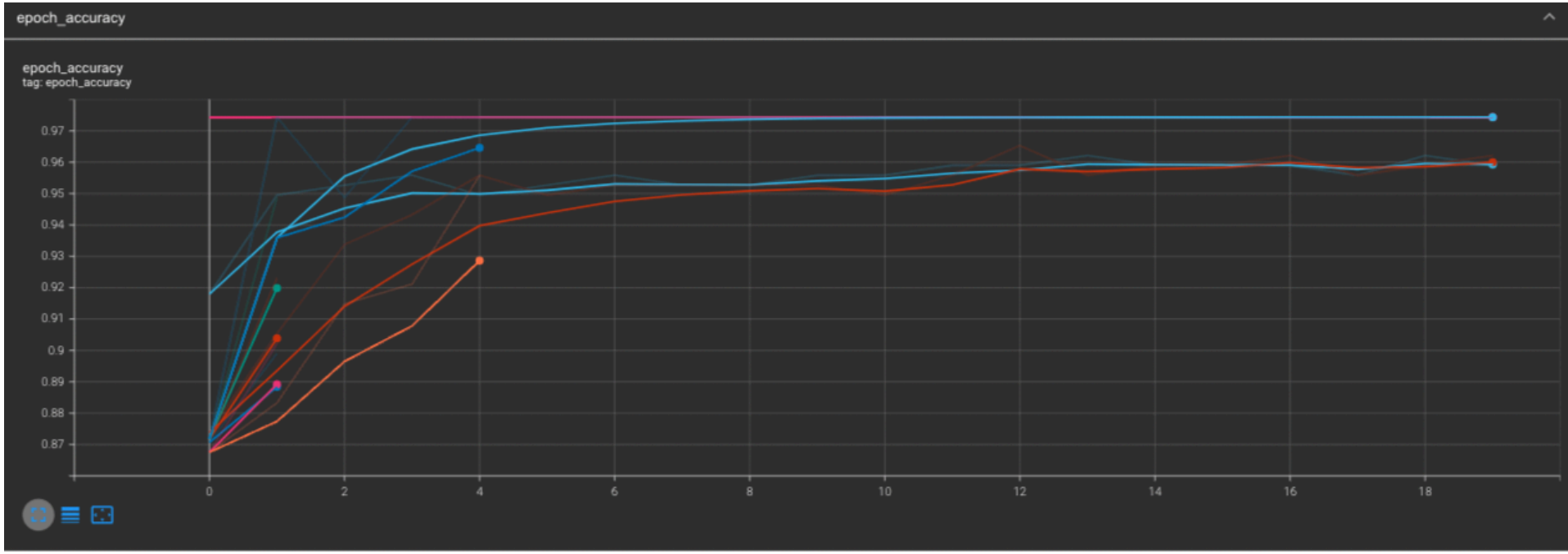



Fig 21: Accuracy trend

# Results - Tensorboard

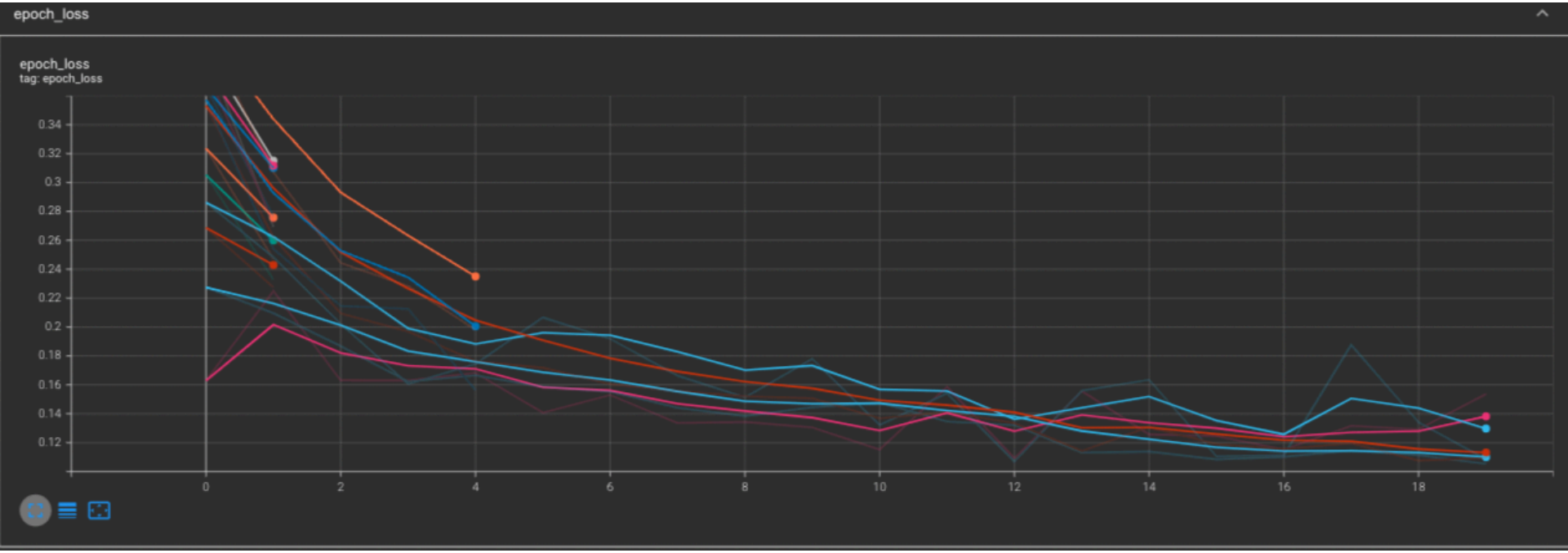



Fig 22: Loss trend

# Results - Test data

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.89 | 0.86 | 0.87 | 36 |
| 1 | 0.29 | 0.33 | 0.31 | 6 |
| accuracy | | | 0.79 | 42 |
| macro avg | 0.59 | 0.60 | 0.59 | 42 |
| weighted avg | 0.80 | 0.79 | 0.79 | 42 |

```
[[31  5]
 [ 4  2]]
```

Fig 23: Test accuracy metrics

# Results - Examples

```
[27]
# predicting images Not flooded class example
img = load_img('output_reshaped/test/10840.jpg', target_size=(img_width, img_height))
x = img_to_array(img)
x = np.expand_dims(x, axis=0)
img.show()
```

```
[25]
images = np.vstack([x])
classes = model.predict(images, batch_size=10)
print(classes)
```

```
...
2022-09-15 00:18:00.278551: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_regis

1/1 [==============================] - 1s 731ms/step
[[0.06705595]]
```

```
[26]
df_test.loc[df_test['filename'] == '10840.jpg']
```

| | filename | label |
|---|---|---|
| 28 | 10840.jpg | 0 |

Fig 24: Non-Flooded example

# Results - Examples

```
# predicting images: Yes flooded class
img = load_img('output_reshaped/test/7325.jpg', target_size=(img_width, img_height))
x = img_to_array(img)
x = np.expand_dims(x, axis=0)
img.show()
```

```
images = np.vstack([x])
classes = model.predict(images, batch_size=10)
print(classes)
```

```
1/1 [==============================] - 0s 69ms/step
[[0.90135497]]
```

```
df_test.loc[df_test['filename'] == '7325.jpg']
```

✓ 0.2s

| | filename | label |
|---|---|---|
| 0 | 7325.jpg | 1 |

Fig 25: Flooded example

# Summary

- Results achieved above 95% accuracy for Train and Validation.
- 80% accuracy for test data.
- Model performed well even for small number of examples using transfer learning.
- Training times low due to reshaping the data.
- Computationally efficient.
- Examples show reliably accuracy in practical settings.

# Future works

- Use non-labeled data for training using Semi-supervised techniques.
- Right now we used only around 400 images, total data available > 2000 HQ images.
- Use max info from the images in further tasks like Semantic segmentation which requires pixel-wise details.
- VQA can also be performed and tested against author's results.

# References


- https://www.climatechange.ai/papers/neurips2021/55
- https://arxiv.org/pdf/2012.02951.pdf
- https://arxiv.org/pdf/2105.08655.pdf
- https://github.com/BinaLab/FloodNet-Challenge-EARTHVISION2021
- https://github.com/sahilkhose/FloodNet
- https://machinelearningmastery.com/how-to-use-transfer-learning-when-developing-convolutional-neural-network-models/
- http://cs231n.stanford.edu/schedule.html