Machine Learning Approaches for Multilingual Multimedia

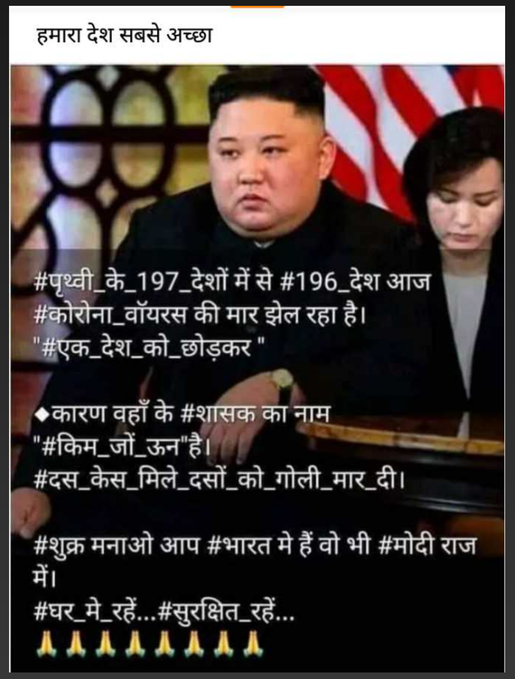
*hereafter known as multimodal ML

Tattle Show and Tell by Kruttika Nadig, 29th July 2020

The WHY

Why does Tattle need multimodal ML?

- Communication (including misinformation) on chat apps & social media spreads via multiple modalities like image, text, video



Why does Tattle need multimodal ML?

- Many Indians are multilingual, which reflects in the content we archive

Why does Tattle need multimodal ML?

- “Meaning” is communicated through combinations of modes and codes

Why does Tattle need multimodal ML?

- Separating modalities and languages before performing ML can lead to information loss
- Need to automate ML tasks regardless of media type & language
- Multimodal ML methodology can be applied to various tasks in our domain (content classification, moderation, search)
- First task - content relevance

The WHOA

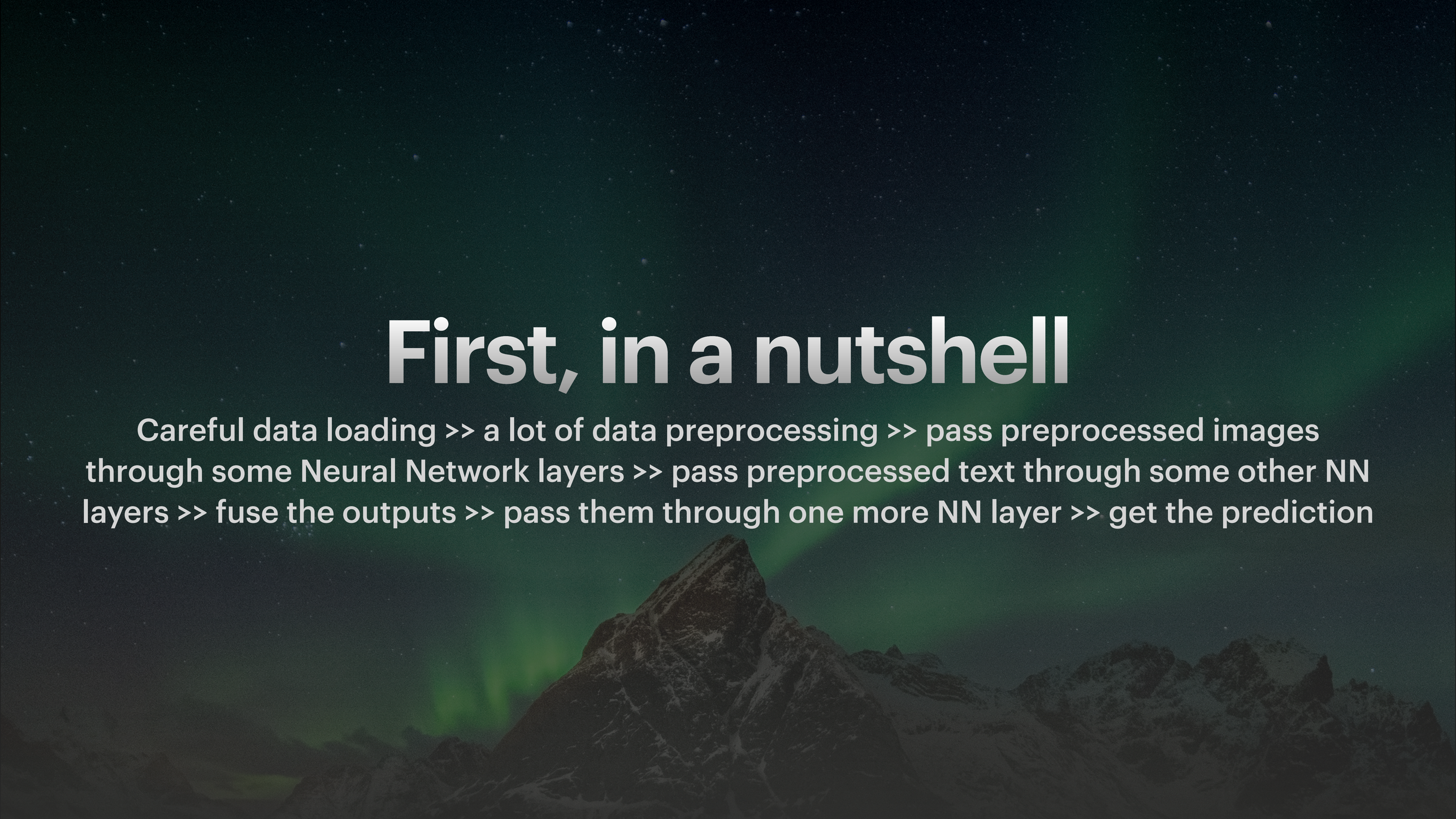
Challenges in multimodal ML

- Less explored area, very few implementation examples
- Different media types need different preprocessing pipelines to make them model-ready
- Involves Computer Vision and Natural Language Processing, both vast fields of their own
- Available NLP resources for Indian languages are limited
- Images on Indian chat apps / social networks are low-quality and noisy, even humans have a hard time categorising them
- Difficult to do with high level Python libraries like Keras, need to use libraries like Pytorch that involve more coding
- Neural networks have low interpretability

The HOW

First, in a nutshell

Careful data loading >> a lot of data preprocessing >> pass preprocessed images through some Neural Network layers >> pass preprocessed text through some other NN layers >> fuse the outputs >> pass them through one more NN layer >> get the prediction



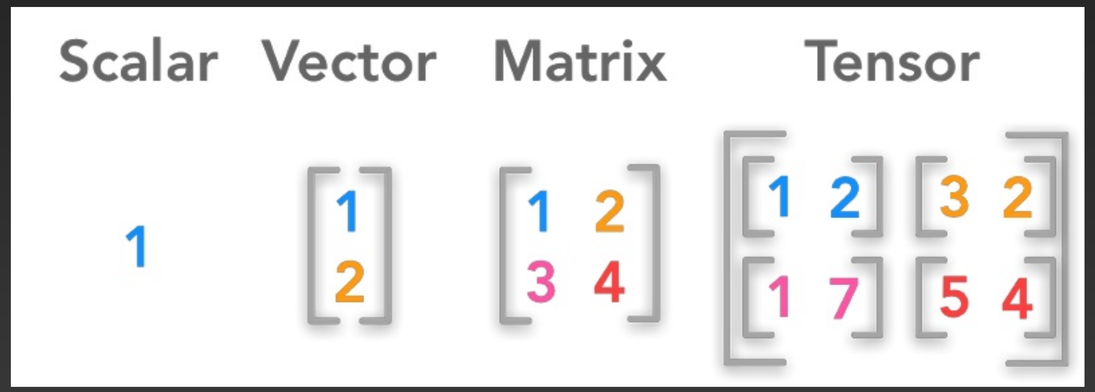
Data loading

- Construct a Pytorch Dataset that will return images, corresponding texts and their labels together
- Pytorch DataLoader allows iterative data loading during model training to avoid loading all data into memory at once

```
class CustomDataset(torch.utils.data.Dataset):
    def __init__(self, img_dir, texts_and_labels, transform):

        with open(texts_and_labels, "r") as fp:
            data = json.load(fp)
        df = pd.DataFrame.from_dict(data, orient="index").reset_index()
        df.columns = ["img_name", "text"]
        df["label"] = df["img_name"].apply(lambda x: 0 if "neg" in x else 1)

        self.img_dir=img_dir
        self.img_names=df["img_name"].values
        self.text = df["text"].values
        self.y = df["label"].values
        self.transform=transform

    def __getitem__(self, idx):
        # For each img_name in img_names, load the corresponding img from the img folder
        img = Image.open(os.path.join(self.img_dir, self.img_names[idx])).convert('RGB')
        # Return the transformed RGB image
        if self.transform is not None:
            img_vector=self.transform(img)

        # Return the corresponding label and text
        label = self.y[idx]
        text = self.text[idx]
        return img_vector,label,text

    def __len__(self):
        return self.y.shape[0]
```

Data preprocessing: images

- Extract text from images
- Transform RGB images into tensors (data containers) of equal size



Data preprocessing: text

- Remove punctuations & convert to lowercase
- Break down sentences into words (tokens)
- Create a vocabulary with indices for each word
- Map the indices to pre-trained word vectors known as embeddings
- Transform sentences into sequences of vocabulary indices
- king + (woman - man) = queen

```
W("woman")-W("man") ≃ W("queen")-W("king")
```

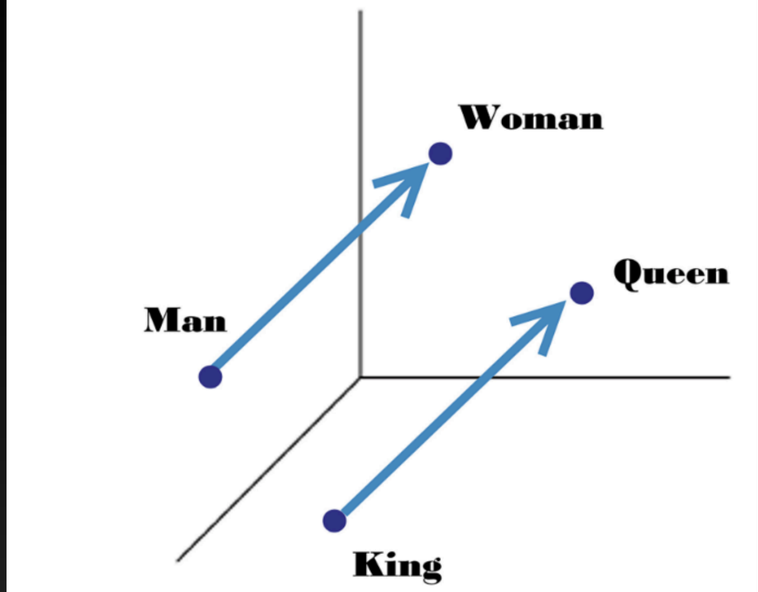
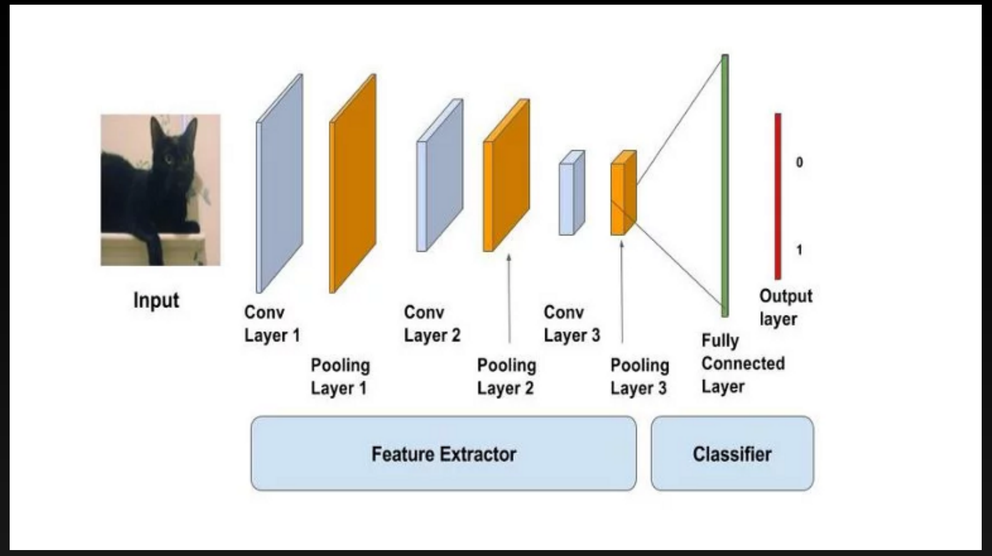


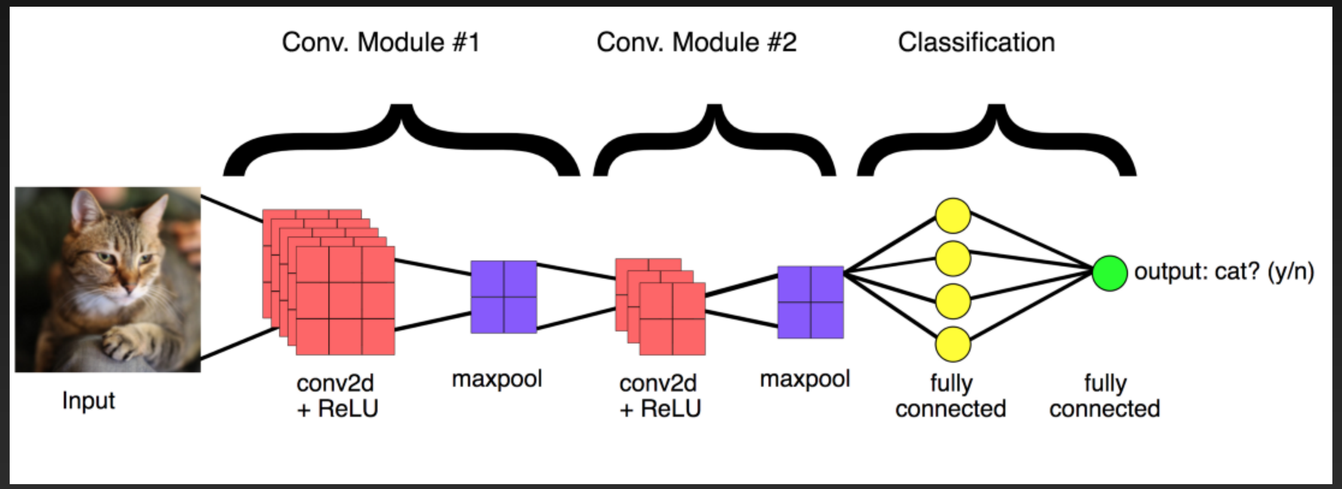
Figure 1. Gender vectors. Source: Lior Shkiller.

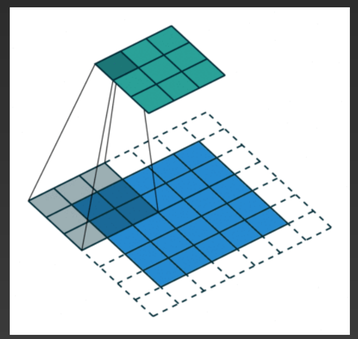
```python
def get_emb_matrix(pretrained, word_counts, emb_size = 300):
    """ Creates training vocabulary, vocab2index and embedding matrix from pretrained word vectors """
    found=0
    not_found=0
    vocab_size = len(word_counts) + 2
    vocab_to_idx = {}
    vocab = ["", "UNK"]
    W = np.zeros((vocab_size, emb_size), dtype="float32")
    W[0] = np.zeros(emb_size, dtype='float32') # adding a vector for padding
    W[1] = np.random.uniform(-0.25, 0.25, emb_size) # adding a vector for unknown words
    vocab_to_idx["UNK"] = 1
    i = 2
    for word in word_counts:
        if word in pretrained:
            W[i] = pretrained[word]
            found+=1
        else:
            W[i] = np.random.uniform(-0.25,0.25, emb_size)
            not_found+=1
        vocab_to_idx[word] = i
        vocab.append(word)
        i += 1
    return W, np.array(vocab), vocab_to_idx, found, not_found
```

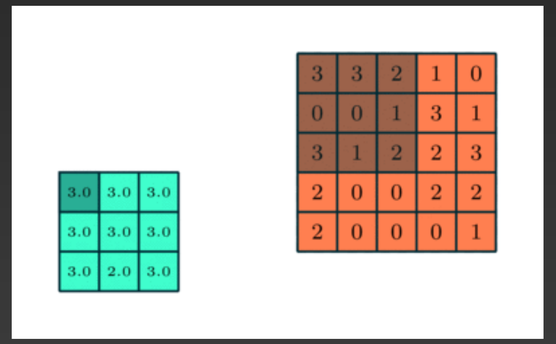
Image representation

- Computer Vision task
- Pass preprocessed images through a CNN (convolutional neural network)
- Extract the second-last layer (the layer before classification) to obtain a learned representation of the image



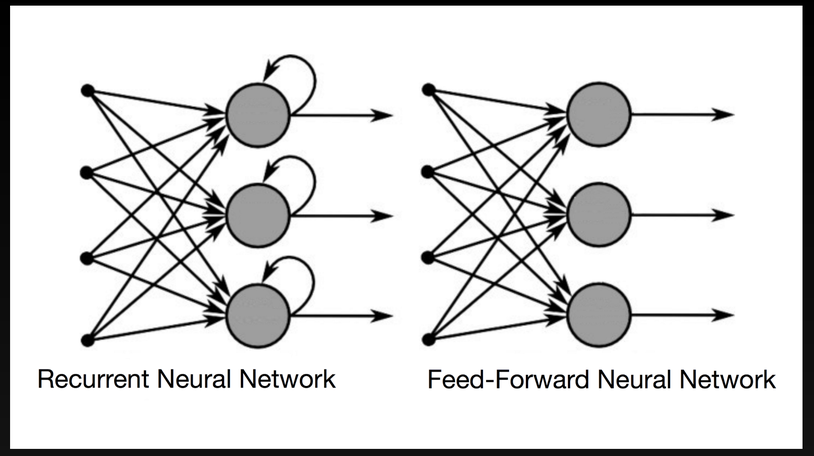


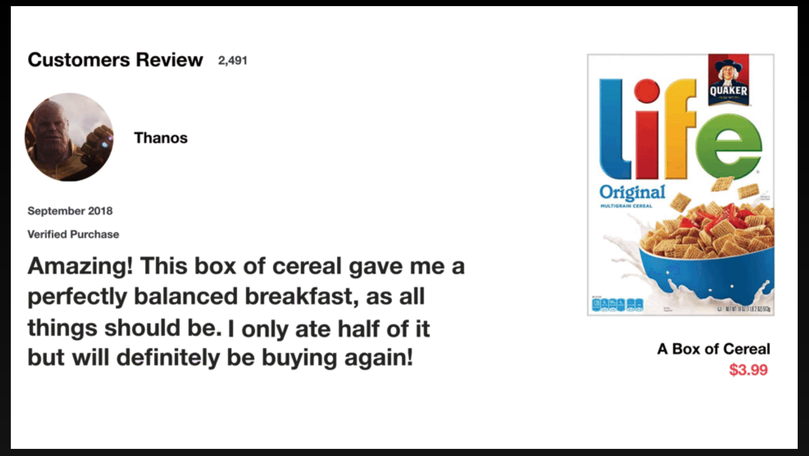


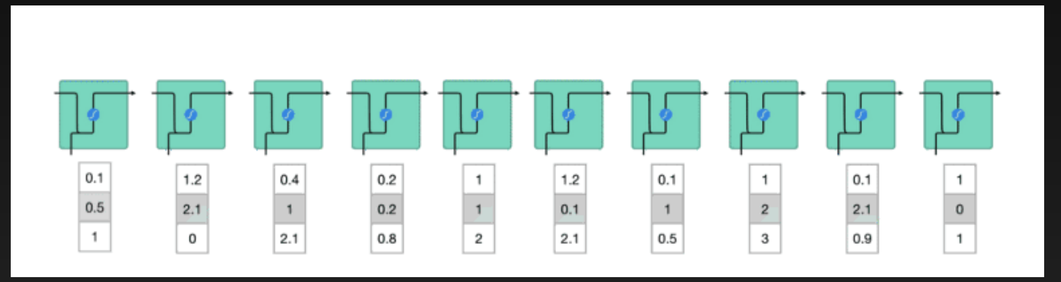


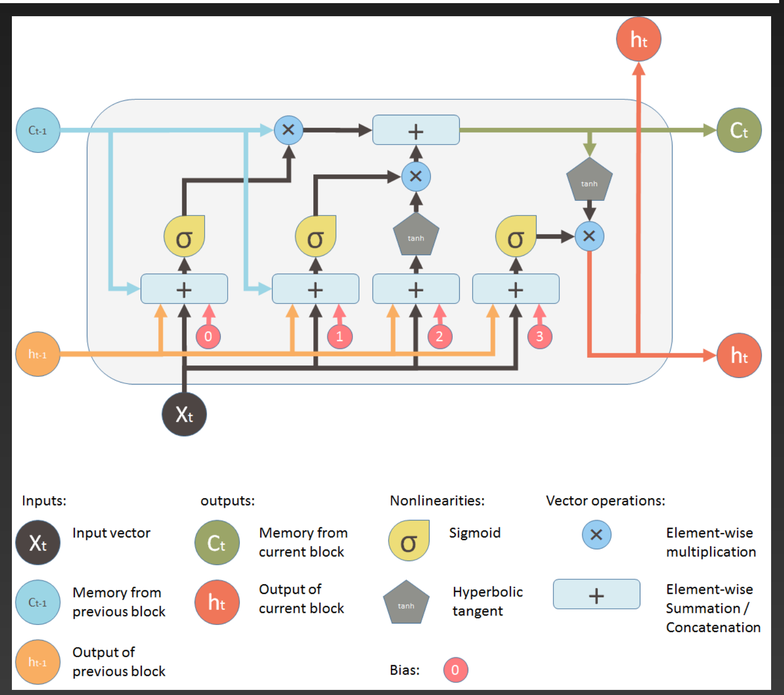
Text representation

- Natural Language Processing task
- Pass preprocessed text through a recurrent neural network like LSTM
- Use transfer learning to get word vectors from pretrained embeddings
- Extract the second-last layer (the layer before classification) to obtain a learned representation of the text









Fusion

- Concatenate image and text representations
- Pass the concatenated representation through a classifier layer to get the predicted probabilities for each class / label
- Technically intermediate or mid-fusion

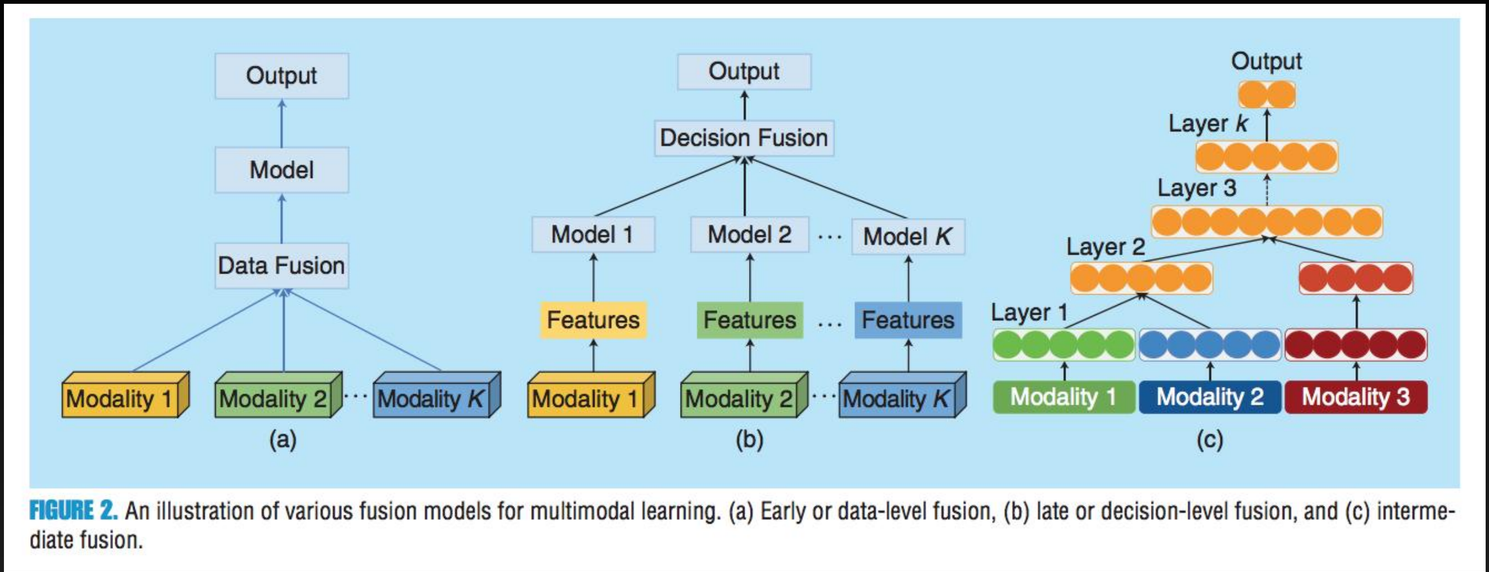
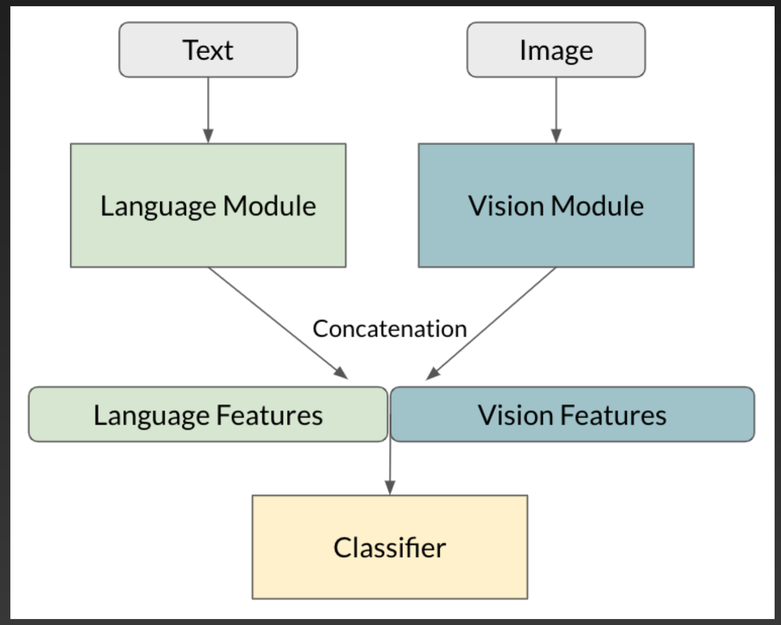


FIGURE 2. An illustration of various fusion models for multimodal learning. (a) Early or data-level fusion, (b) late or decision-level fusion, and (c) intermediate fusion.



Model training

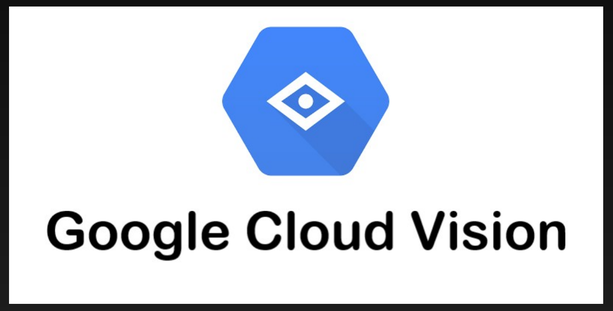
- Define hyperparameters like number of epochs, learning rate, loss function
- Pass the training and validation DataLoaders to the model, iteratively train each batch of data till loss reaches a minimum
- Convert probabilities to predictions and calculate accuracy
- Inference: pass test DataLoader to model and make predictions
- Current accuracy: ~80% with minimal hyperparameter tuning (per class accuracy 78% and 83% respectively)

Train model

```
def train_model(model, epochs, lr, train_loader, train_seq_loader):
    parameters = filter(lambda p: p.requires_grad, model.parameters())
    optimizer = torch.optim.Adam(parameters, lr=lr)
    model.train()
    for i in range(epochs):
        sum_loss = 0.0
        total = 0
        for i, data in enumerate(zip(train_loader, train_seq_loader)):
            img_vectors, text_sequences, y_train = data[0][0], data[1], data[0][1]
            optimizer.zero_grad()
            y_pred = model(img_vectors, text_sequences)
            loss = F.cross_entropy(y_pred, y_train)
            loss.backward()
            optimizer.step()
            sum_loss += loss.item()*y_train.shape[0]
            total += y_train.shape[0]
        val_loss, val_acc = validation_metrics(model, validation_loader, valid_seq_loader)
        print("train loss %.3f, val loss %.3f, val accuracy %.3f" % (sum_loss/total, val_loss, val_acc))

def validation_metrics (model, validation_loader=validation_loader, valid_seq_loader=valid_seq_loader):
    # Initialize the prediction and label lists(tensors)
    predlist=torch.zeros(0,dtype=torch.long, device='cpu')
    lbllist=torch.zeros(0,dtype=torch.long, device='cpu')
    model.eval()
    correct = 0
    total = 0
    sum_loss = 0.0
    for i, data in enumerate(zip(validation_loader, valid_seq_loader)):
        #
        img_vectors, text_sequences, y_valid = data[0][0], data[1], data[0][1]
        y_hat = model(img_vectors, text_sequences)
        loss = F.cross_entropy(y_hat, y_valid)
        #print("y_hat:",y_hat)
        pred = torch.max(y_hat, 1)[1]
        print("pred:",pred)
        # Append batch prediction results
        predlist=torch.cat([predlist,pred.view(-1).cpu()])
        lbllist=torch.cat([lbllist,y_valid.view(-1).cpu()])
        # Calculate accuracy
```

Our tech stack for multimodal ML

- Python for pipeline development
- Pytorch (a Python library) for building and training neural networks
- Google Vision API for extracting text and other features from images (paid tool)
- Fasttext word embeddings by Facebook AI for finding relationships between multilingual words
- ResNet neural network architecture by Facebook AI for learning patterns in images
- MongoDB and Amazon S3 for data storage



The WAY

Further experiments & improvements (1/2)

Data engineering

- Archive-level text extraction

Model accuracy

- Hyperparameter tuning
- Use transformers

Further experiments & improvements (2/2)

Scope

- Integrate videos into pipeline
- Include more languages

R&D

- Project images & text into same vector space
- Train on more datasets like FB hate speech images, Whatsapp Indian election images

External sources for media used in this presentation

- https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21
- https://nerdthecoder.wordpress.com/2019/02/03/recurrent-neural-net/
- https://www.oreilly.com/content/capturing-semantic-meanings-using-deep-learning/
- https://ieeexplore.ieee.org/document/8103116/metrics#metrics
- https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53
- https://towardsdatascience.com/introduction-to-convolutional-neural-network-cnn-de73f69c5b83
- https://towardsdatascience.com/covolutional-neural-network-cb0883dd6529

Thanks!