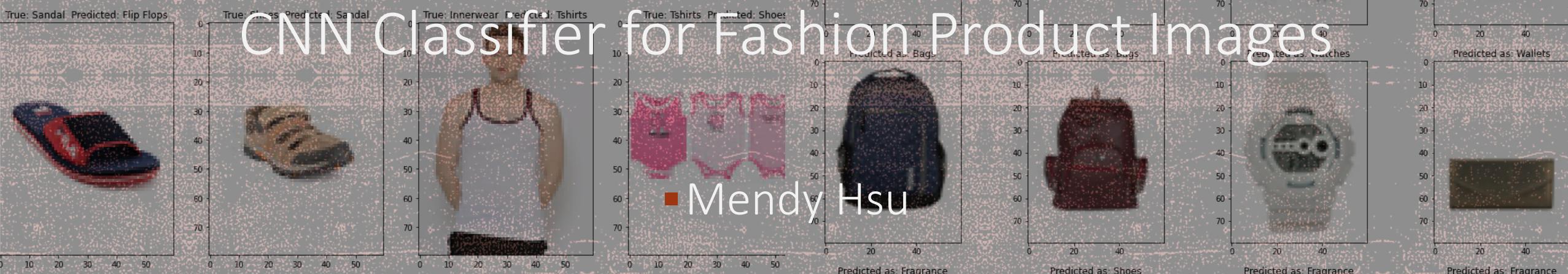


True: Sandal Predicted: Flip Flops

IDENTIFY FASHION PRODUCT TYPE BASED ON ITS IMAGE



Classify the fashion products. Why? Who would care?

- Fashion e-commerce accounted for roughly [29.5 percent of the total fashion retail sales in the United States \(2020\).](#)
- One of main challenges for e-commerce - **categorizing/labeling the products from the images.**
- Many businesses have developed and advanced their algorithms to classify/label products further into **multi-categories/features** (ex: the product's color, texture or pattern, style, or look).
- Build the **product recommendation system** based on the product features.
- Well-known e-commences specializing in customizing or matching fashion products to customers: Stitch Fix, DAILYLOOK, and Nordstrom's Trunk Club for fashion appeals; Birchbox for personal care products, to name a few.

Problem Formulation

- Image classification poses an exciting computer vision puzzle.
- Building a classification model for fashion product images would be an excellent start to dive into deep neural networks.
- Project goals:
 - To deal with the **imbalanced fashion image dataset**.
 - Build **Convolutional Neural Networks (CNN)** for image classification with **Keras API**.
 - Evaluate the model performances.

Data Information

- Image labels: **styles.csv**
 - **Number of fields:** **10**
(id, masterCategory, subCategory, articleType, etc.)
 - **Number of rows:** **44440** hourly records.
- Image files: **44440 images** (80-by-60)
(image_id*.jpg).

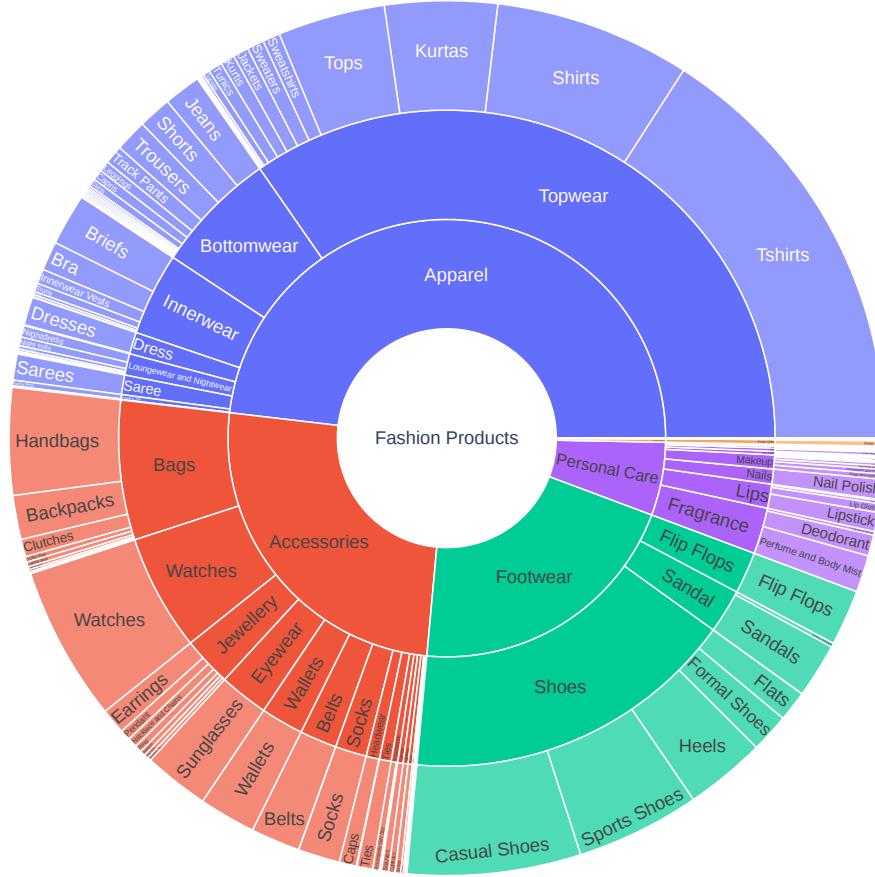
Data Wrangling & Feature Engineering

- Deal with the Imbalanced Labels
 - Defined a new categorical feature - **Class**
It combines the three hierarchical labels (`masterCategory`, `subCategory`, `articleType`).
 - **Class** has **30** product labels.

Data Wrangling & Feature Engineering

- Reassign Class to Images Under the Ambiguous Category (Free Items)
 - For each mislabeled item, compute **Text Similarity** (cosine similarity) between its `productDisplayName` and the **30 labels**.
 - Among the 30 labels, the maximal one should be assigned to the product.

Fashion Product Category



Fashion Product Category (reassigned)



Proportion of Categorical Labels

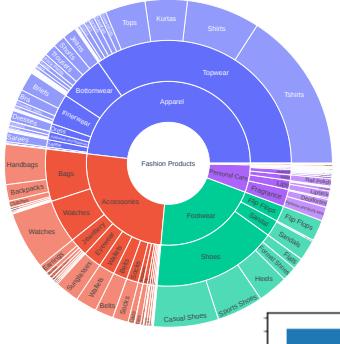
General Balance Measurement for Categorical Variable

$$p_i = \frac{n_i}{N}, \quad N = \sum_{i=1}^C n_i, \quad H = -\sum_{i=1}^C p_i \log p_i, \quad H_{max} = \log C$$

$$Balance = \frac{H}{H_{max}} = -\frac{1}{\log C} \sum_{i=1}^C \frac{n_i}{N} \log \frac{n_i}{N},$$

N : the total number of samples across all classes.
 n_i : sample count of class i .

Categorical Feature	Measure of Balance	Sample Distribution
masterCategory	0.671	Figure 1
subCategory	0.644	Figure 2
articleType	0.731	Figure 3
Class	0.815	Figure 5



Product Counts (Original Categorical Labels)

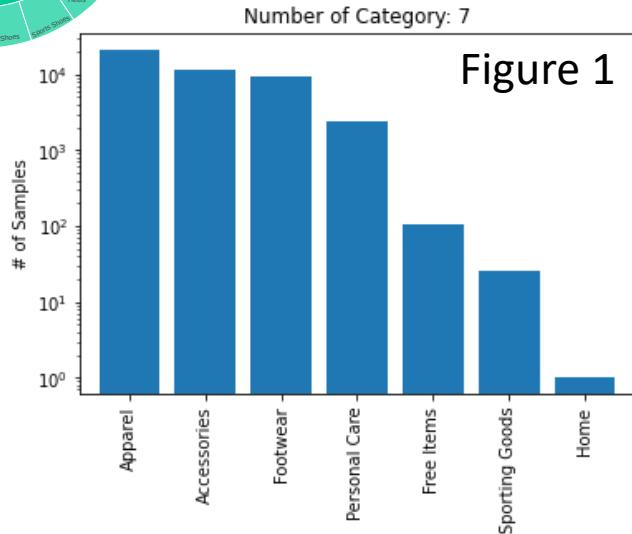


Figure 1

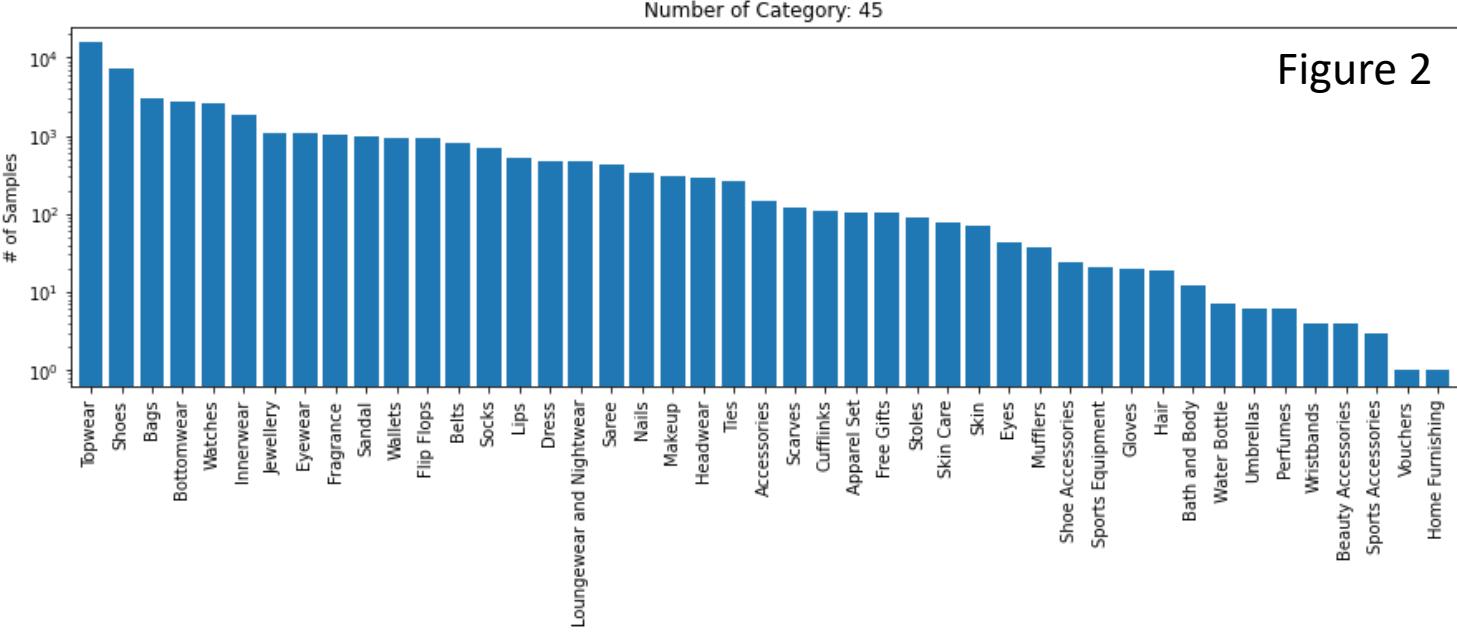
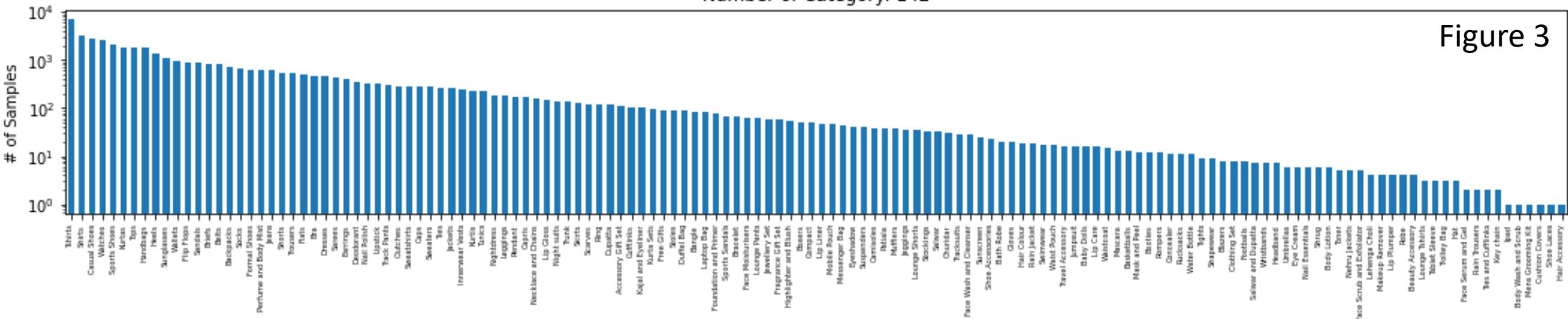
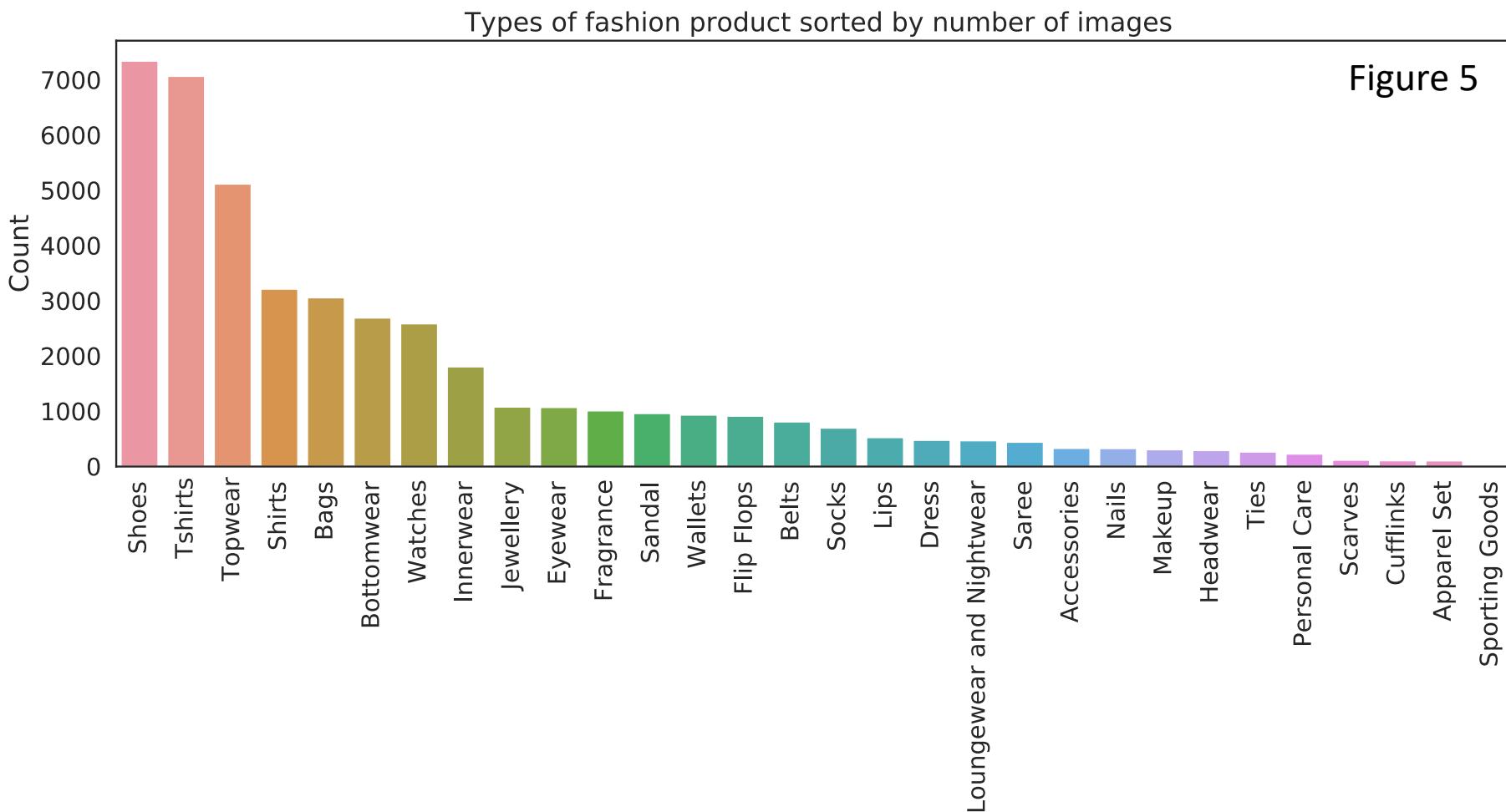


Figure 2





Product Counts (Refined Categorical Labels)



Pre-processing and Training Data Development

- **Scale the Pixel Values**

- 3 color channels, in the range of 0 to 255
→ **rescale by 1/255** to transform the pixel values to range of 0-1, as preferred by the neural network models.
- Also, Keras API provides option to train with color (default setting) or grayscale images.

- **Image Augmentation**

- **Horizontally flipping** randomly on images.
- Most of the pictures are well centered and correctly orientated. Applying shifts or rotations would not improve the model performances.

- **Split Train-validation and Hold-on Sets**

- **Training (72%); validation (18%); hold-on (10%).**
- Set a **batch size of 32**.
Each of the train and validation datasets are divided into groups of 32 images.

```
## Build the model using the Keras API
chanDim = -1 # RGB Channel is last
model = tf.keras.models.Sequential()

# 1st CONV & RELU => BatchNorm => POOL layer set
model.add(Conv2D(32, (3, 3), activation='relu', padding='same'
                 input_shape=inputShape))
model.add(BatchNormalization(axis=chanDim))
model.add(Conv2D(32, (3, 3), activation='relu', padding='same')
model.add(BatchNormalization(axis=chanDim))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))

# 2nd CONV & RELU => BatchNorm => POOL layer set
model.add(Conv2D(64, (3, 3), activation='relu', padding='same')
model.add(BatchNormalization(axis=chanDim))
model.add(Conv2D(64, (3, 3), activation='relu', padding='same')
model.add(BatchNormalization(axis=chanDim))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))

# first (and only) set of FC(fully connected) => RELU layers
model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.25))

# softmax classifier
model.add(Dense(classes, activation='softmax'))
```

CNN Model Architectures

- Input → Convo (ReLU) → BatchNorm
- Convo (ReLU) → BatchNorm
- Max Pooling 1 → Dropout (0.2)

- Convo (ReLU) → BatchNorm
- Convo (ReLU) → BatchNorm
- Max Pooling 2 → Dropout (0.2)

- FC layer (ReLU) → BatchNorm
- Dropout (0.25)
- Output (Softmax)

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 80, 60, 32)	896
batch_normalization (BatchNo	(None, 80, 60, 32)	128
conv2d_1 (Conv2D)	(None, 80, 60, 32)	9248
batch_normalization_1 (Batch	(None, 80, 60, 32)	128
max_pooling2d (MaxPooling2D)	(None, 40, 30, 32)	0
dropout (Dropout)	(None, 40, 30, 32)	0
conv2d_2 (Conv2D)	(None, 40, 30, 64)	18496
batch_normalization_2 (Batch	(None, 40, 30, 64)	256
conv2d_3 (Conv2D)	(None, 40, 30, 64)	36928
batch_normalization_3 (Batch	(None, 40, 30, 64)	256
max_pooling2d_1 (MaxPooling2	(None, 20, 15, 64)	0
dropout_1 (Dropout)	(None, 20, 15, 64)	0
flatten (Flatten)	(None, 19200)	0
dense (Dense)	(None, 512)	9830912
batch_normalization_4 (Batch	(None, 512)	2048
dropout_2 (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 30)	15390

Total params: 9,914,686

Trainable params: 9,913,278

Non-trainable params: 1,408

CNN Model Architectures

Input → Convo (ReLU) → BatchNorm
→ Convo (ReLU) → BatchNorm
→ Max Pooling 1 → Dropout (0.2)

→ Convo (ReLU) → BatchNorm
→ Convo (ReLU) → BatchNorm
→ Max Pooling 2 → Dropout (0.2)

→ FC layer (ReLU) → BatchNorm
→ Dropout (0.25)
→ Output (Softmax)