CNN Implementation

Convolutional Neural Network on Fashion MNIST Dataset

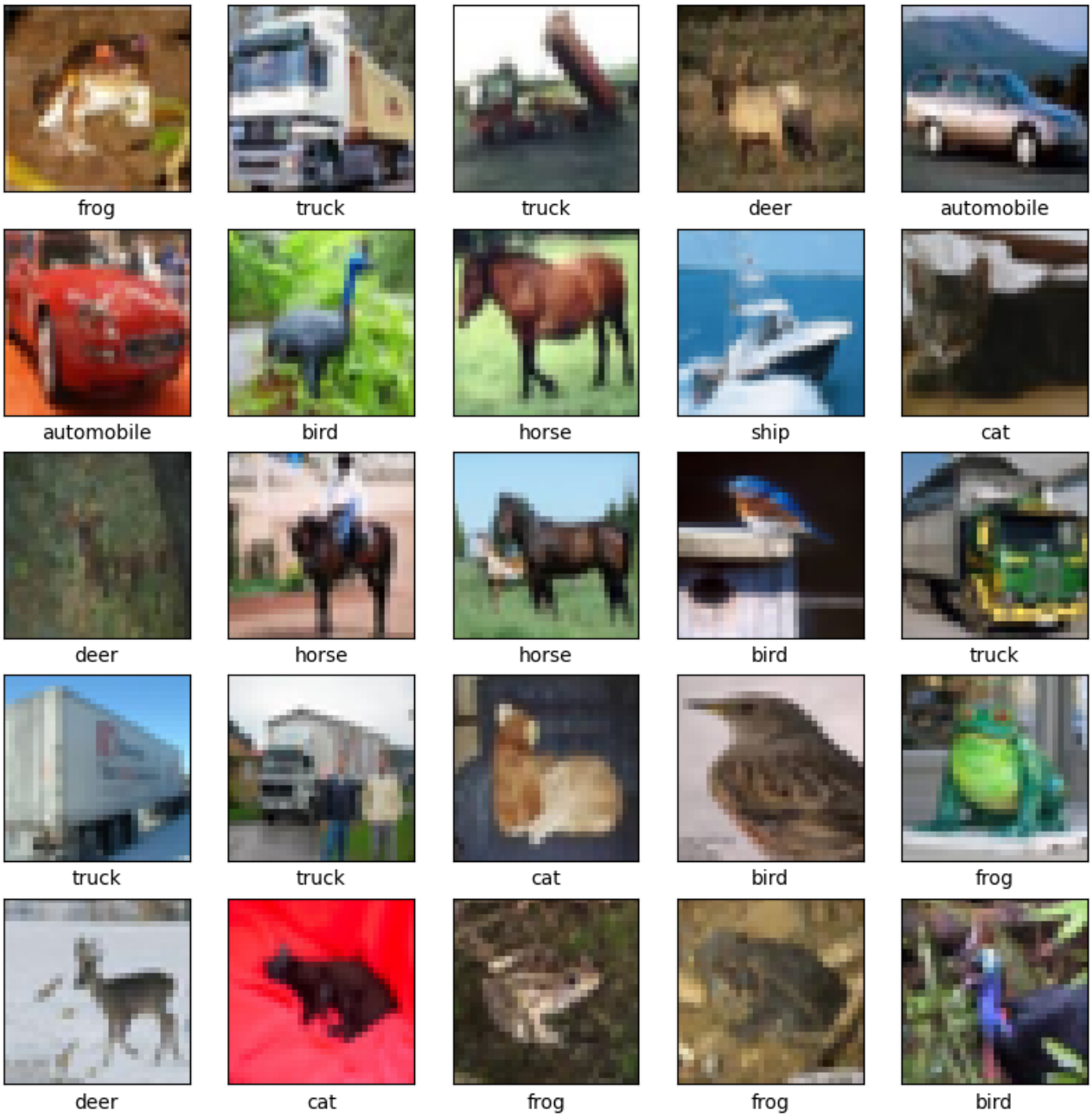
Author M.M.Mirzaie

July 30, 2025

Contents

1 Introduction

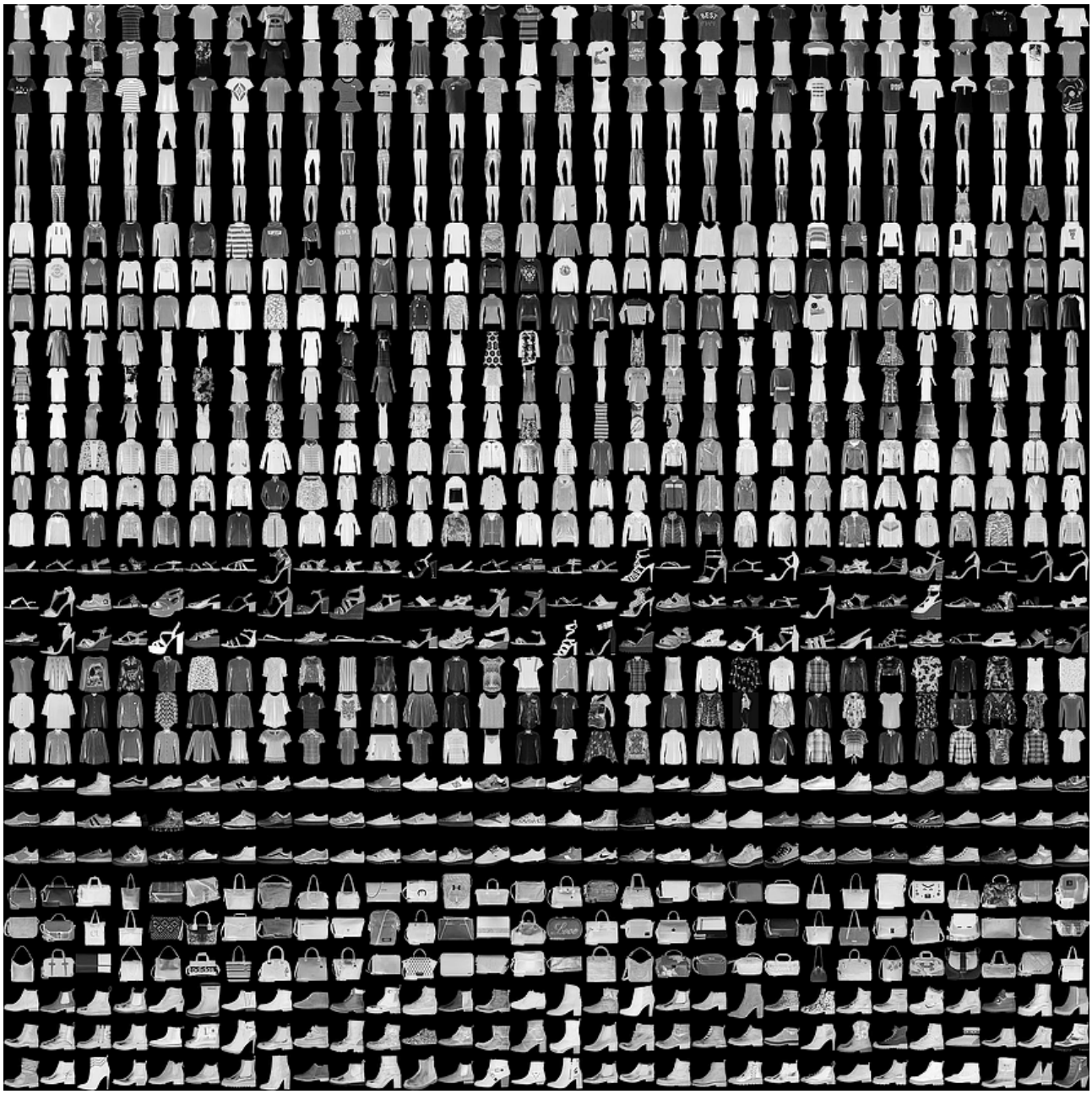
A convolutional neural network (CNN) is a category of machine learning model. Specifically, it is a type of deep learning algorithm that is well suited to analyzing visual data. CNNs are commonly used to process image and video tasks. And, because CNNs are so effective at identifying objects, they are frequently used for computer vision tasks, such as image recognition and object recognition, with common use cases including self-driving cars, facial recognition and medical image analysis.



2 Dataset Description

Fashion-MNIST is a dataset of Zalando's article images—consisting of a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28x28 grayscale image, associated with a label from 10 classes. We intend Fashion-MNIST to serve as a direct drop-in replacement for the original MNIST dataset for benchmarking machine learning algorithms. It shares the same image size and structure of training and testing splits.

Here's an example of how the data looks (each class takes three-rows):



2.1 Fashion-MNIST Classes

The 10 classes are as follows:

- 0 - Tshirt/top
- 1 - Trouser
- 2 - Pullover
- 3 - Dress
- 4 - Coat
- 5 - Sandal
- 6 - Shirt
- 7 - Sneaker
- 8 - Bag
- 9 - Ankle boot

Each image is 28 pixels in height and 28 pixels in width, for a total of 784 pixels in total.

Each pixel has a single pixel-value associated with it, indicating the lightness or darkness of that pixel, with higher numbers meaning darker.The training and test data sets have 785 columns.

The first column consists of the class labels (see above), and represents the article of clothing. The rest of the columns contain the pixel-values of the associated image.

To locate a pixel on the image, suppose that we have decomposed x as x = i * 28 + j, where i and j are integers between 0 and 27. The pixel is located on row i and column j of a 28 x 28 matrix. For example, pixel31 indicates the pixel that is in the fourth column from the left, and the second row from the top, as in the ascii-diagram below.

3 Import Dataset and Libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from tensorflow.keras.datasets import fashion_mnist
(x_train,y_train),(x_test,y_test) = fashion_mnist.load_data()

```

We are using TensorFlow fashion MNIST dataset.

3.1 Shape

x is the 28x28 images.

```
x_train.shape

```

(60000, 28, 28)

```
x_test.shape

```

(10000, 28, 28)

y is the target_output(lables) of the images.

```
y_train.shape

```

(60000,)

```
y_test.shape

```

(10000,)

3.2 Going throw each item

```
    item.max()
    item.min()

```

Pixel-value is an integer between 0 and 255.

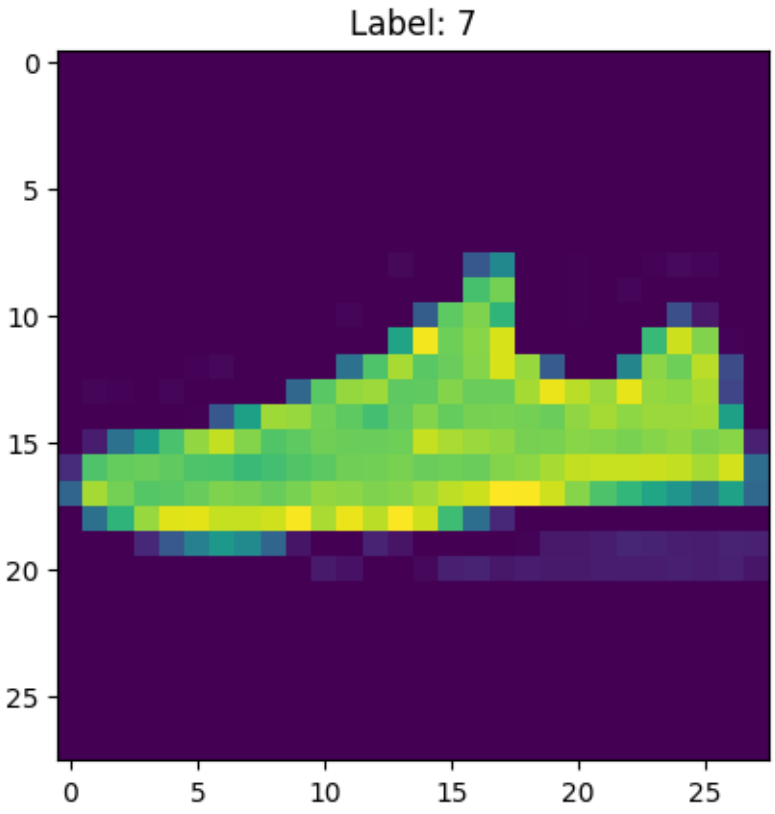
np.uint8(254)
np.uint8(0)

```
    item = x_train[np.random.randint(0, 6000)]
    item.shape

```

Every time you run this code a random picture of dataset is saved in item.

```
    item

```


```
    plt.imshow(item)
    plt.title(f"Label: {y_train[randNumber,]}")

```



3.3 Convert Integer Labels Into One-Hot Encoded Vectors

```
from tensorflow.keras.utils import to_categorical

y_cat_train = to_categorical(y_train)
y_cat_test = to_categorical(y_test)
```

```
y_cat_train[0]
```

array([0., 0., 0., 0., 0., 0., 0., 0., 0., 1.])
So its from Ankle boot class.

```
y_cat_train.shape
```

(60000, 10)

```
y_cat_test[0]
```

array([0., 0., 0., 0., 0., 0., 0., 0., 0., 1.])

```
y_cat_test.shape
```

(10000, 10)

4 Normalization

4.1 What is Normalization?

Normalization is the process of transforming data to a common scale, typically [0, 1] or [-1, 1], without altering the relative relationships between values. It's used to ensure that features contribute equally to a machine learning model, especially when they originally have different ranges or units.

4.2 X_train, X_test Normalization

```
#Normalize the data
x_train = x_train/255
x_test = x_test/255
```

```
x_train[0].max()
```

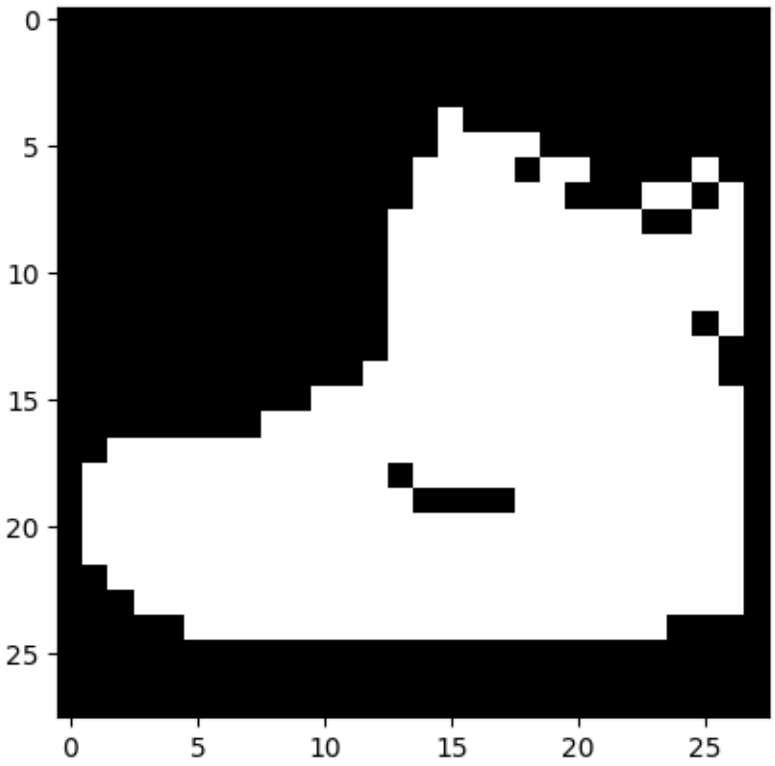
np.float64(1.0)

```
x_train[0].min()
```

np.float64(0.0)

4.3 Binary Image

```
binary_image = (x_train[0] > 0.5).astype(np.uint8)

plt.imshow(binary_image, cmap='gray')

```



4.4 Add Channel Dimension for CNN Compatibility

CNNs, especially in frameworks like TensorFlow/Keras, expect input data in a specific format: (batch_size, height, width, channels) so we add channels to the dataset.

- Samples: Number of images (e.g., 60,000 for x_train in MNIST).
- Height: Number of pixels vertically (e.g., 28 for MNIST).
- Width: Number of pixels horizontally (e.g., 28 for MNIST).
- Channels: Number of color or feature planes:
  - Grayscale images: 1 channel (intensity values, as in MNIST).
  - RGB images: 3 channels (red, green, blue).
  - Other cases: More channels for specialized data (e.g., hyperspectral images).

```
    x_test = x_test.reshape(10000,28,28,1)
    x_train = x_train.reshape(60000,28,28,1)

```

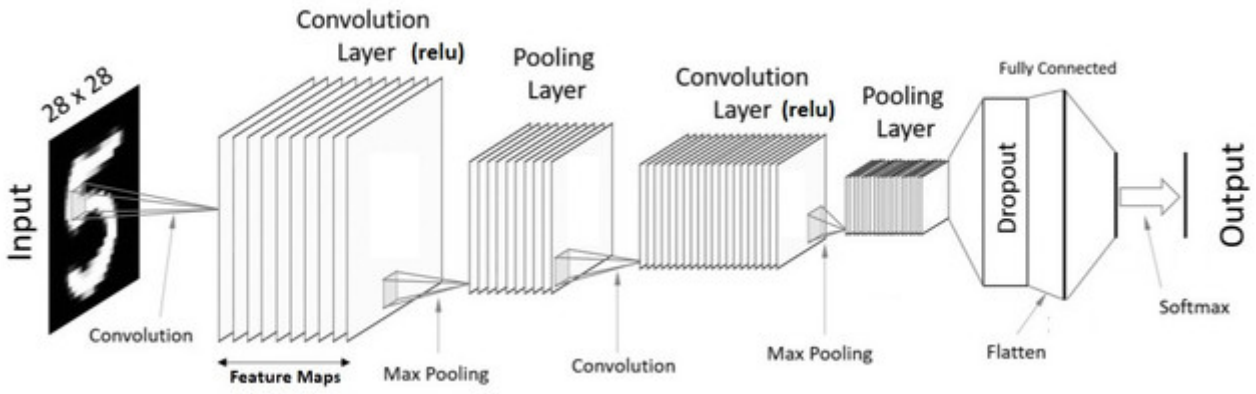
```
    x_train.shape

```

(10000, 28, 28, 1)

```
    x_test.shape

```

(60000, 28, 28, 1)

5 Build The Model




```
#Build the model
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, MaxPool2D, Flatten, Dropout

```

(10000, 28, 28, 1)

5.1 Conv2D Layer

Applies 2D convolution to extract features (e.g., edges, patterns) from the input images.

- filters=32: Creates 32 feature maps by applying 32 different filters (3x3 kernels). Each filter detects different patterns (e.g., horizontal edges, vertical edges).
- kernel_size=(3,3): Each filter is a 3x3 matrix that slides over the image, computing dot products to produce feature maps.
- activation='linear': Applies a linear activation (i.e., no transformation, $f(x) = x$). This is unusual since activations like ReLU are typically applied directly in Conv2D. Instead, you apply LeakyReLU separately (next line).
- input_shape=(28,28,1): Specifies the input shape for the first layer: 28x28 pixels, 1 channel (grayscale, as discussed in your reshape to (60000, 28, 28, 1) for CNN compatibility).
- padding='same': Adds zero-padding around the input image so the output feature maps have the same spatial dimensions (28x28) as the input. Without padding (padding='valid'), the output would shrink (e.g., to 26x26 for a 3x3 kernel).
- .BatchNormalization(): Normalizes the outputs of the previous layer (Conv2D) to stabilize and accelerate trainin

Output Shape: (batch_size, 28, 28, 32) (32 feature maps, each 28x28, due to padding='same').

```
model = Sequential()

model.add(Conv2D(filters=32,kernel_size=(3,3),activation = 'linear',input_shape
=(28,28,1),padding='same'))
model.add(tf.keras.layers.BatchNormalization())


```

5.1.1 Add LeakyReLU(alpha = 0.1)

- Applies the LeakyReLU activation function to introduce non-linearity, allowing the model to learn complex patterns.
- alpha=0.1 means negative inputs are scaled by 0.1 (e.g., -1 becomes -0.1) instead of being set to 0 (as in standard ReLU).
- Dropout(0.25): Regularizes the model by randomly deactivating 25% of neurons during training to prevent overfitting.

Output Shape: (batch_size, 28, 28, 32)

```
1
2     model.add(tf.keras.layers.LeakyReLU(alpha = 0.1))
3     model.add(Dropout(0.25))
4
```

5.1.2 MaxPool2D Layer

Reduces the spatial dimensions of the feature maps (downsampling) while retaining important features.

- pool_size=(2,2): Takes the maximum value in each 2x2 region of the feature map, reducing height and width by half (e.g., 28x28 to 14x14).
- padding='same': Adds padding to ensure the output dimensions are rounded up (e.g., 14x14 instead of 13x13 if the input size isn't perfectly divisible).

Output Shape: (batch_size, 14, 14, 32) (height and width halved, 32 filters unchanged).

```
1     model.add(MaxPool2D(pool_size=(2,2),padding='same'))
2
```

5.2 Conv2D Layer 2

Similar to the first block, but with 64 Filters

```
1
2   model.add(Conv2D(64,(3,3),activation = 'linear',padding = 'same'))
3   model.add(tf.keras.layers.BatchNormalization())
4   model.add(tf.keras.layers.LeakyReLU(alpha = 0.1))
5   model.add(Dropout(0.25))
6   model.add(MaxPool2D(pool_size=(2,2),padding='same'))
7
8
```

5.3 Conv2D Layer 3

128 Filters

```
1
2   model.add(Conv2D(128,(3,3),activation = 'linear',padding = 'same'))
3   model.add(tf.keras.layers.BatchNormalization())
4   model.add(tf.keras.layers.LeakyReLU(alpha = 0.1))
5   model.add(Dropout(0.25))
6   model.add(MaxPool2D(pool_size=(2,2),padding='same'))
7
8
```

5.4 Dense Layers

5.4.1 What Are Dense Layers?

Dense layers, also called fully connected layers, are neural network layers where every input is connected to every output neuron through a learnable weight. In your CNN, the dense layers follow the Flatten layer and are responsible for making the final classification decision (e.g., identifying whether an input image is a digit "5" or another digit in the MNIST dataset).

5.4.2 Why Convert 3D Feature Maps to a 1D Vector?

The Flatten layer is a critical bridge between the convolutional/pooling layers (which operate on 2D/3D spatial data) and the dense (fully connected) layers (which expect 1D input vectors).

Without Flatten, passing (batch_size, 4, 4, 128) to a Dense layer would cause an error in Keras, as it cannot process multi-dimensional inputs

```
1
2  model.add(Flatten())
3  model.add(Dense(128,activation = 'linear'))
4  model.add(tf.keras.layers.BatchNormalization())
5  model.add(tf.keras.layers.LeakyReLU(alpha = 0.1))
6  model.add(Dropout(0.25))
7
```

And in the end we have the Dense(10) layer for classification 10 MNIST classes.

```
1  model.add(Dense(10,activation = 'softmax'))
2
```

6 Model Summary

```
1
2   model.summary()
3
```

Model: "sequential_7"

Layer (type)	Output Shape	Param #
conv2d_19 (Conv2D)	(None, 28, 28, 32)	320
batch_normalization_8 (BatchNormalization)	(None, 28, 28, 32)	128
leaky_re_lu_25 (LeakyReLU)	(None, 28, 28, 32)	0
dropout_13 (Dropout)	(None, 28, 28, 32)	0
max_pooling2d_18 (MaxPooling2D)	(None, 14, 14, 32)	0
conv2d_20 (Conv2D)	(None, 14, 14, 64)	18,496
batch_normalization_9 (BatchNormalization)	(None, 14, 14, 64)	256
leaky_re_lu_26 (LeakyReLU)	(None, 14, 14, 64)	0
dropout_14 (Dropout)	(None, 14, 14, 64)	0
max_pooling2d_19 (MaxPooling2D)	(None, 7, 7, 64)	0
conv2d_21 (Conv2D)	(None, 7, 7, 128)	73,856
batch_normalization_10 (BatchNormalization)	(None, 7, 7, 128)	512
leaky_re_lu_27 (LeakyReLU)	(None, 7, 7, 128)	0
dropout_15 (Dropout)	(None, 7, 7, 128)	0
max_pooling2d_20 (MaxPooling2D)	(None, 4, 4, 128)	0
flatten_6 (Flatten)	(None, 2048)	0
dense_12 (Dense)	(None, 128)	262,272
batch_normalization_11 (BatchNormalization)	(None, 128)	512
leaky_re_lu_28 (LeakyReLU)	(None, 128)	0
dropout_16 (Dropout)	(None, 128)	0
dense_13 (Dense)	(None, 10)	1,290

7 Model Compilation

Configures the model for training by specifying the loss function, optimizer, and evaluation metrics.

```
model.compile(loss = 'categorical_crossentropy', optimizer= 'adam', metrics = ['
  accuracy'])
```

- Loss Function: categorical_crossentropy Loss $= -\sum_{i=1}^{10} y_{\text{true},i} \log(y_{\text{pred},i})$
  - y_{true}: One-hot label (e.g., [0, 0, 0, 0, 0, 1, 0, 0, 0, 0] for digit 5).
  - y_{pred}: Predicted probabilities from softmax (e.g., [0.01, 0.02, ..., 0.80, ...]).
- Optimizer: adam Updates the model's weights (in Conv2D and Dense layers) to minimize the loss using the Adam (Adaptive Moment Estimation) algorithm.
- Metrics: ['accuracy'] Tracks the fraction of correct predictions during training and validation.

```
#Train the model
history = model.fit(x_train, y_cat_train, batch_size= 64, epochs = 3,
validation_data=(x_test, y_cat_test))
```

model.fit(): This is the core function in Keras used to train a model. It takes the training data and corresponding labels, and iteratively adjusts the model's internal weights to minimize the loss function and improve accuracy.

batch size: the number of training examples that will be processed together in each batch during training. The model's weights are updated after processing each batch.

epochs: This parameter determines how many times the training process will iterate over the entire training dataset.

8 Model Validation

8.1 Accuracy and Loss

Print Loss and accuracy with evaluate function.

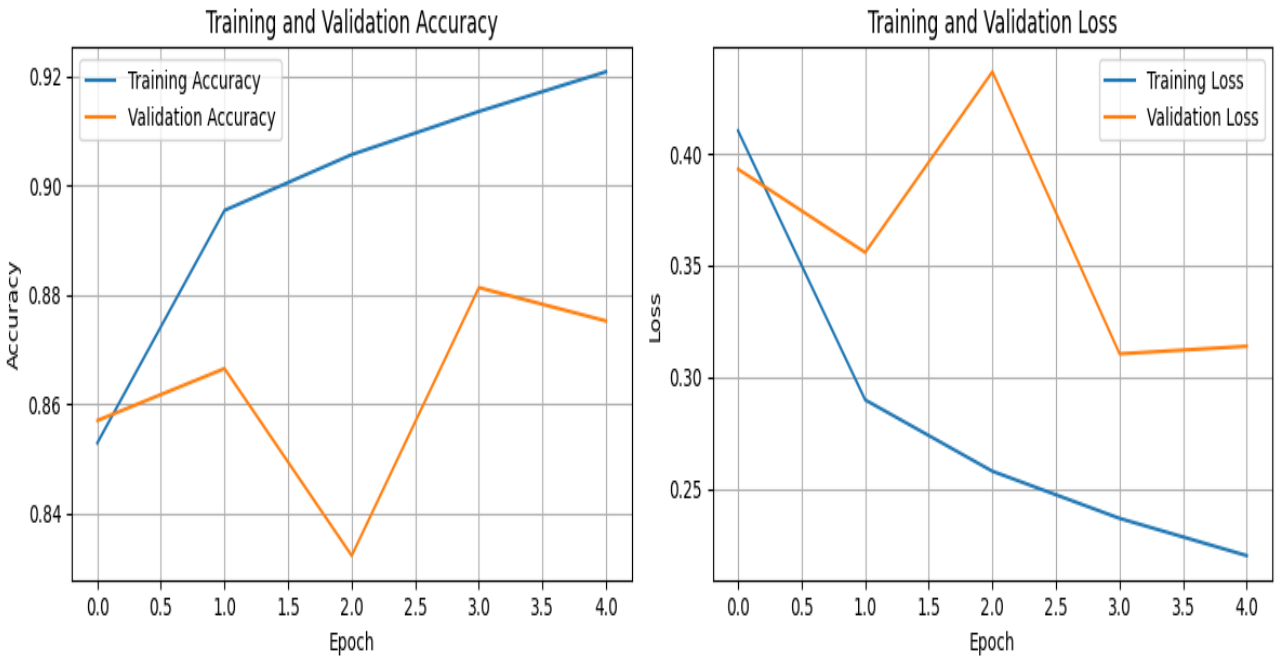
```
1
2     test_loss, test_acc = model.evaluate(x_test, y_cat_test, verbose=2)
3     print(f'\nTest accuracy: {test_acc:.4f}')
4
```

```
313/313 - 6s - 19ms/step - accuracy: 0.8752 - loss: 0.3138

Test accuracy: 0.8752
```

8.2 Accuracy and Loss Base on Epochs Plot

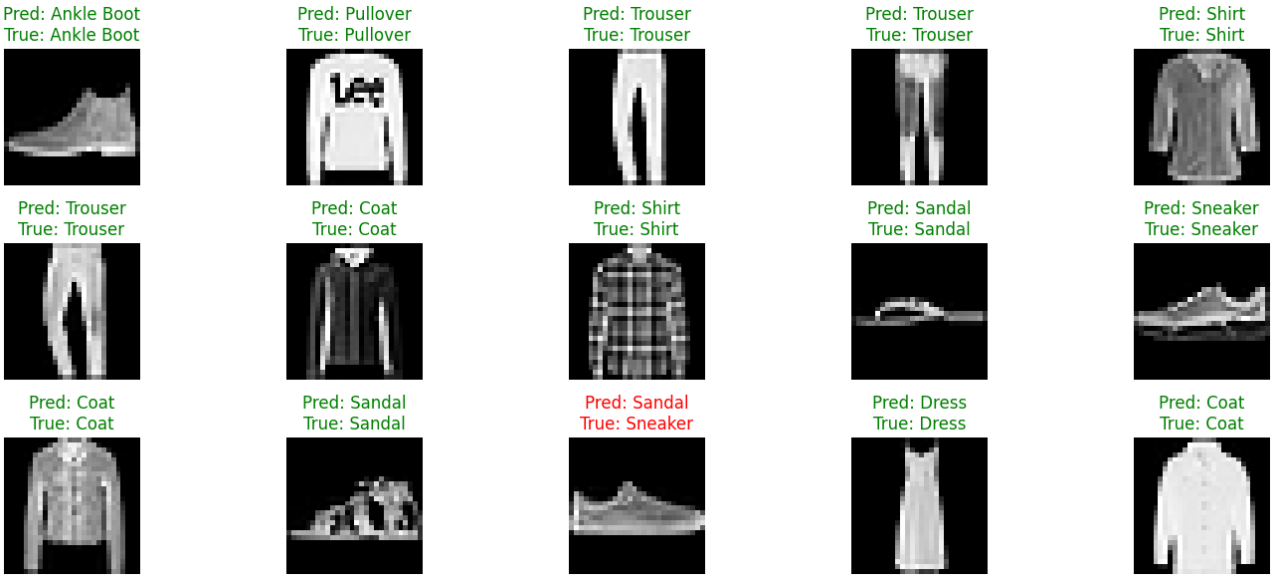
Using History Class to show plot of Accuracy and Loss Base on Epochs Plot.

```
    # Get the number of epochs from the history
    epochs = range(len(history.history['accuracy']))
    plt.figure(figsize=(12, 4))
    # Plot Training and Validation Accuracy
    plt.subplot(1, 2, 1)
    plt.plot(epochs, history.history['accuracy'], label='Training Accuracy')
    plt.plot(epochs, history.history['val_accuracy'], label='Validation Accuracy')
    plt.title('Training and Validation Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.grid(True)
    # Plot Training and Validation Loss
    plt.subplot(1, 2, 2)
    plt.plot(epochs, history.history['loss'], label='Training Loss')
    plt.plot(epochs, history.history['val_loss'], label='Validation Loss')
    plt.title('Training and Validation Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()
    plt.grid(True)
    plt.tight_layout()
    plt.show()

```



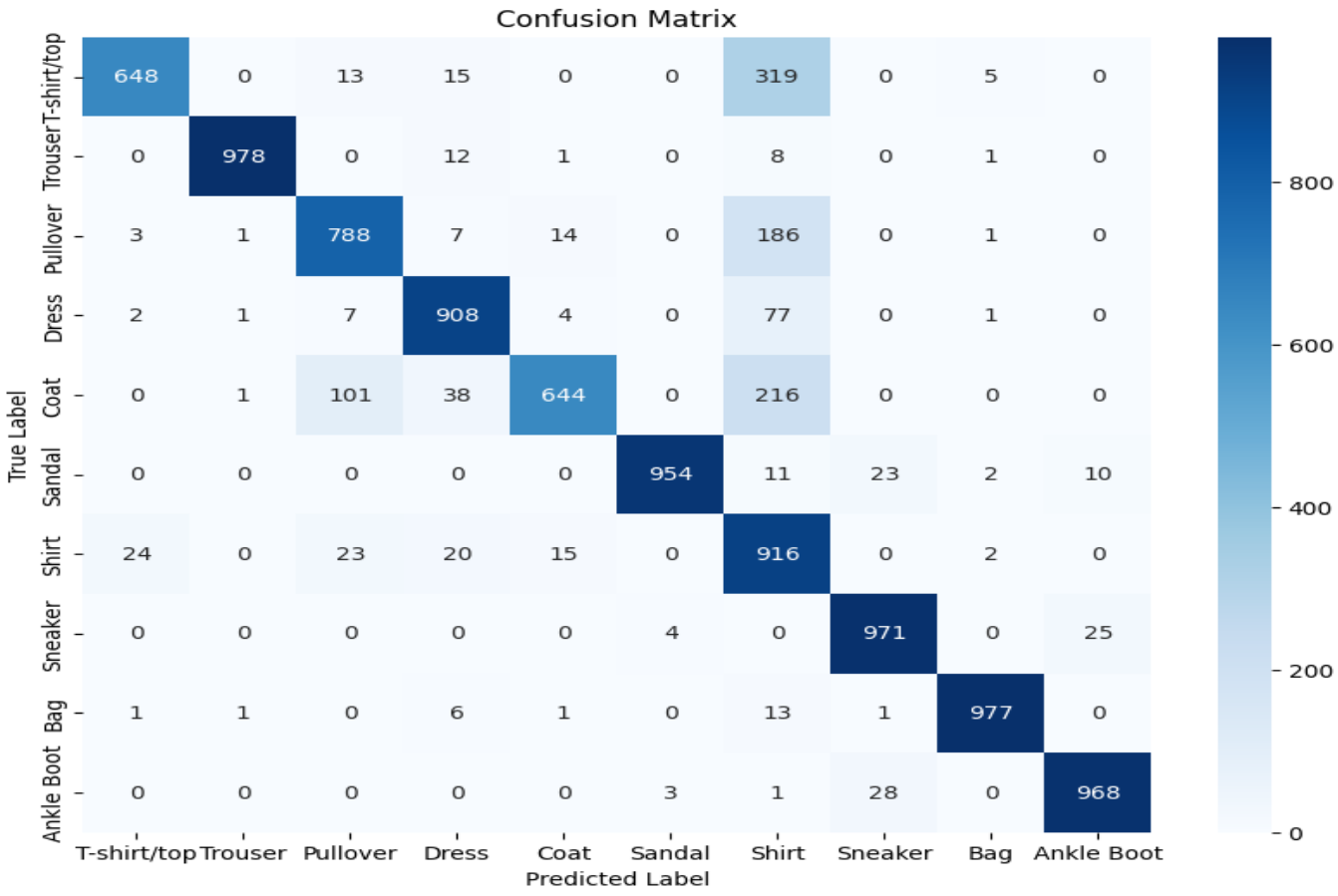
8.3 True Or False Prediction

```
# Make predictions on the test images
predictions = model.predict(x_test)

# Display some test images with predicted labels
plt.figure(figsize=(15, 6))
for i in range(15):
plt.subplot(3, 5, i+1)
# Display the image
plt.imshow(x_test[i].reshape(28, 28), cmap='gray')

# Get predicted and true labels
pred_label_num = np.argmax(predictions[i])
true_label_num = y_test[i] # Use y_test for true labels

# Get label names using the labels dictionary
pred_label_name = labels[pred_label_num]
true_label_name = labels[true_label_num]

# Determine title color based on prediction correctness
color = 'green' if pred_label_num == true_label_num else 'red'

# Set the title with predicted and true label names
plt.title(f"Pred: {pred_label_name}\nTrue: {true_label_name}", color=color)
plt.axis('off') # Hide axes

plt.tight_layout() # Adjust layout to prevent titles/labels from overlapping
plt.show()


```



8.4 Confusion Matrix

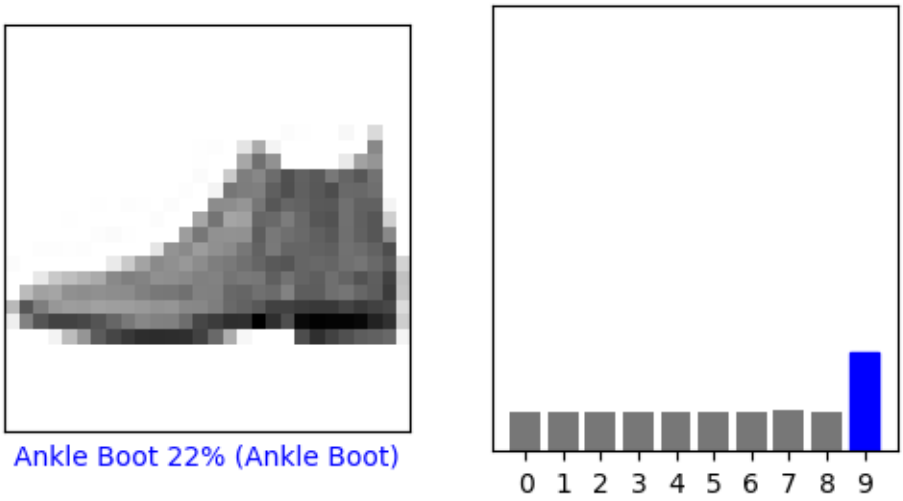
```
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns

# Get true and predicted labels
y_true = y_test
y_pred = np.argmax(predictions, axis=1)

# Confusion Matrix
plt.figure(figsize=(10, 8))
cm = confusion_matrix(y_true, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
xticklabels=labels.values(), yticklabels=labels.values()) # Use label names for ticks
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()

# Classification Report
print("\nClassification Report:")
print(classification_report(y_true, y_pred, target_names=labels.values(), digits=4)) # Use label names in report


```



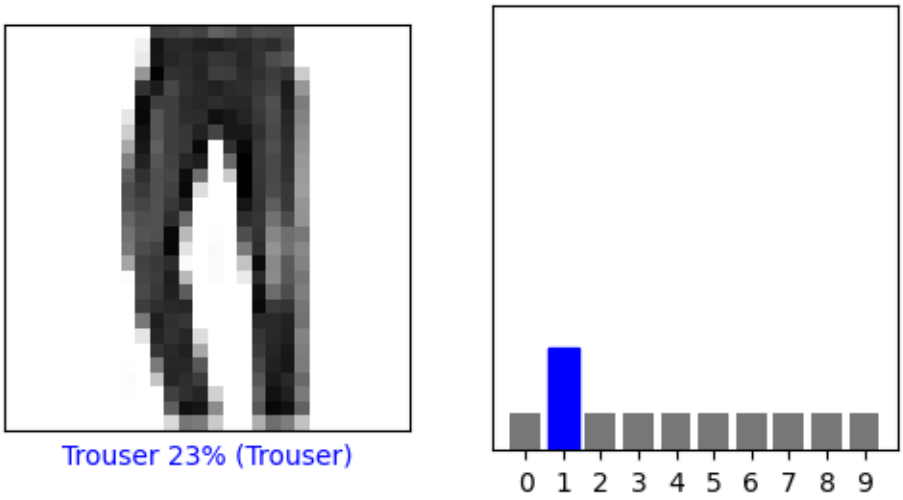
8.5 Model's Prediction

This plot visualizes the model's prediction for a test image, showing the image itself with the predicted and true labels (highlighted in red for incorrect predictions), and a bar chart displaying the model's confidence scores across all 10 classes, with the predicted and true class probabilities highlighted.

```
import matplotlib.pyplot as plt
import numpy as np

def plot_image(i, predictions_array, true_label, img):
true_label, img = true_label[i], img[i]
plt.grid(False)
plt.xticks([])
plt.yticks([])

plt.imshow(img, cmap=plt.cm.binary)

predicted_label = np.argmax(predictions_array)
if predicted_label == true_label:
color = 'blue'
else:
color = 'red'

plt.xlabel("{} {:2.0f}% ({})".format(labels[predicted_label],
100*np.max(predictions_array),
labels[true_label]),
color=color)

def plot_value_array(i, predictions_array, true_label):
true_label = true_label[i]
plt.grid(False)
plt.xticks(range(10))
plt.yticks([])
thisplot = plt.bar(range(10), predictions_array, color="#777777")
plt.ylim([0, 1])
predicted_label = np.argmax(predictions_array)

thisplot[predicted_label].set_color('red')
thisplot[true_label].set_color('blue')

probability_model = tf.keras.Sequential([model,
tf.keras.layers.Softmax()])
predictions = probability_model.predict(x_test)

i = 0
plt.figure(figsize=(6,3))
plt.subplot(1,2,1)
plot_image(i, predictions[i], y_test, x_test)
plt.subplot(1,2,2)
plot_value_array(i, predictions[i],  y_test)
plt.show()


```




```
    i = 5 # Choose a different index
plt.figure(figsize=(6,3))
plt.subplot(1,2,1)
plot_image(i, predictions[i], y_test, x_test)
plt.subplot(1,2,2)
plot_value_array(i, predictions[i],  y_test)
plt.show()


```



Generates a classification report for your model's performance on the test data. It provides metrics like precision, recall, and f1-score for each class, along with the overall accuracy. This helps you understand how well your model is performing for each category of fashion item.

```
from sklearn.metrics import classification_report

num_classes = 10
target_names = [f"Class {i}" for i in range(num_classes)]

print(classification_report(y_test, predicted_classes, target_names =
target_names))


```

	precision	recall	f1-score	support
Class 0	0.94	0.59	0.72	1000
Class 1	0.98	0.97	0.98	1000
Class 2	0.93	0.57	0.71	1000
Class 3	0.92	0.82	0.87	1000
Class 4	0.82	0.68	0.75	1000
Class 5	1.00	0.85	0.92	1000
Class 6	0.44	0.91	0.59	1000
Class 7	0.86	0.96	0.91	1000
Class 8	0.98	0.96	0.97	1000
Class 9	0.95	0.96	0.96	1000
accuracy			0.83	10000
macro avg	0.88	0.83	0.84	10000
weighted avg	0.88	0.83	0.84	10000

9 Upload Your Picture

With the code blow you can upload your picture and see how accurate it will be predict.

```
import cv2
from google.colab import files

# Function to preprocess external images
def preprocess_external_image(img_path):
img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
#img = cv2.bitwise_not(img)  # Inverting colors (optional)
img = cv2.resize(img, (28, 28))
img = img.astype('float32') / 255
return img.reshape(1, 28, 28, 1)

# 3. Upload image
uploaded = files.upload()
img_path = list(uploaded.keys())[0]

# 4. Predict and display result
img = preprocess_external_image(img_path)
prediction = model.predict(img)
predicted_num = np.argmax(prediction)
confidence = np.max(prediction)

# Get the predicted label name using the labels dictionary
predicted_label_name = labels[predicted_num]

plt.imshow(img.reshape(28, 28), cmap='gray')
plt.title(f'Predicted Label: {predicted_label_name} (Confidence: {confidence:.2%})'
  )
plt.axis('off')
plt.show()

print(f'\nFinal Result: Predicted label {predicted_label_name} with confidence {
  confidence:.2%}')


```

10 Refrences

- https://github.com/zalandoresearch/fashion-mnist
- https://github.com/guilhermedom/cnn-fashion-mnist
- https://www.kaggle.com/code/kanncaa1/convolutional-neural-network-cnn-tutorial
- https://www.kaggle.com/code/faressayah/fashion-mnist-classification-using-cnns
- https://www.kaggle.com/code/gpreda/cnn-with-tensorflow-keras-for-fashion-mnist
- www.tensorflow.org
- www.techtarget.com
- https://grok.com