AI Arabic Sign Recognition

Made by: Ahmed Hossam 20100867

Presented to: Eng. Yasmina

Inroduction

AI-based Arabic Sign Language recognition using images involves the use of computer vision techniques to automatically recognize and interpret Arabic sign language gestures from still images. In this approach, images are captured of individuals performing various sign language gestures, with each image representing a specific word or phrase. The goal is to train machine learning models to identify and classify these gestures accurately.

The process typically includes preprocessing steps such as resizing and normalizing the images, followed by training a Convolutional Neural Network (CNN) or other image classification models to extract relevant features like hand shape, position, and orientation. Once trained, the model can predict the corresponding Arabic word or phrase from a given image of a sign language gesture.

This technology has great potential to improve communication accessibility for the deaf and hard-of-hearing communities in Arabic-speaking countries, enabling real-time translation and interaction through image-based recognition.

Related work

The field of Arabic Sign Language recognition has evolved significantly, from traditional machine learning methods to deep learning approaches. With the advent of large, labeled datasets and advanced neural network architectures, deep learning models, particularly CNNs, have become the go-to method for static sign language gesture recognition. These systems are improving accessibility and communication for the Arabic-speaking deaf community. However, challenges remain in creating models that can accurately handle complex gestures, different sign language variations, and the integration of additional cues such as facial expressions. As research progresses, hybrid and multimodal approaches

offer promising directions for improving the accuracy and applicability of sign language recognition systems.

Proposed Models

HandCraftedCNN Model

- **input image** is passed through the convolutional layers, where it undergoes filtering to extract features.
- **Pooling** reduces the spatial dimensions of the feature maps.
- The feature maps are **flattened** into a 1D vector.
- The flattened vector is passed through **fully connected layers** to make the final prediction.
- **Dropout** is applied during training to prevent overfitting.
- Finally, the network outputs a **probability distribution** over the classes, and the class with the highest probability is selected as the model's prediction.

The **HandCraftedCNN** model is a relatively simple yet effective convolutional neural network designed to classify images (e.g., Arabic sign language gestures). It extracts relevant features from the images using convolutional layers, reduces the size of the feature maps using pooling, and then makes a final classification using fully connected layers. The use of dropout ensures that the model generalizes well and does not overfit to the training data.

```python
class HandCraftedCNN(nn.Module):
    def __init__(self, num_classes, input_size=(224, 224)):  # input_size is now (224, 224)
        super(HandCraftedCNN, self).__init__()
        self.conv1 = nn.Conv2d(3, 32, kernel_size=3, stride=1, padding=1)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1)
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)
        self.dropout = nn.Dropout(0.5)

        # Dynamically compute the flattened size after convolutional layers
        conv_output_size = self._get_conv_output_size(input_size)
        self.fc1 = nn.Linear(conv_output_size, 512)
        self.fc2 = nn.Linear(512, num_classes)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(x.size(0), -1)  # Flatten
        x = F.relu(self.fc1(x))
        x = self.dropout(x)
        x = self.fc2(x)
        return x
```

ResNet-18 Model

ResNet (Residual Networks) is a type of deep CNN that introduces the concept of **residual learning**. The main idea is to allow the network to learn residual mappings instead of direct mappings, making it easier to train very deep networks by mitigating the vanishing gradient problem.

ResNet-18 is a relatively shallow version of the ResNet architecture, with only 18 layers. Despite its relatively small depth compared to other ResNet variants, ResNet-18 has proven to be very effective for many image classification tasks.

- **Input Image**: The input image is passed into the model.
- **Feature Extraction**: The image is processed through the convolutional layers of ResNet-18, which extracts increasingly complex features from the image.
- **Residual Learning**: During this process, residual connections allow the model to learn mappings more effectively.
- **Final Classification**: After feature extraction, the image is classified into one of the `num_classes` categories by the final fully connected layer.
- **Output**: The output is a set of class scores (one for each possible class), and the class with the highest score is selected as the prediction.

The `ResNetClassifier` is based on the ResNet-18 architecture, which is known for its use of residual connections to train deeper networks effectively. The model is initialized with pretrained weights and then customized by replacing the final classification layer to fit a new task (e.g., Arabic sign language recognition). The image is passed through the

network, features are extracted, and the final classification is made based on the learned representations. This approach leverages the power of pretrained networks while customizing them to specific problems with a relatively simple modification to the final layer.

```
class ResNetClassifier(nn.Module):
    def __init__(self, num_classes):
        super(ResNetClassifier, self).__init__()
        # Load pretrained ResNet-18 model
        self.resnet = models.resnet18(pretrained=True)
        # Replace the final fully connected layer
        self.resnet.fc = nn.Linear(self.resnet.fc.in_features, num_classes)

    def forward(self, x):
        return self.resnet(x)
```

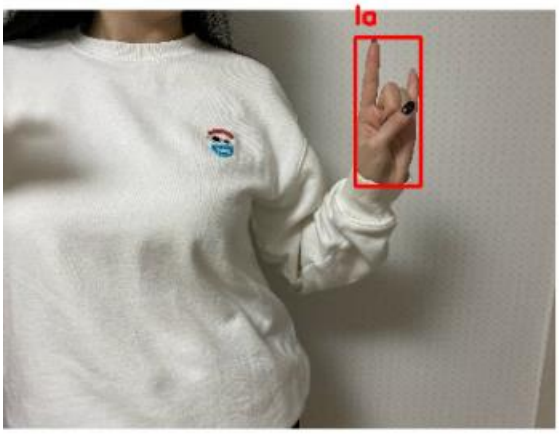
Experimental Work

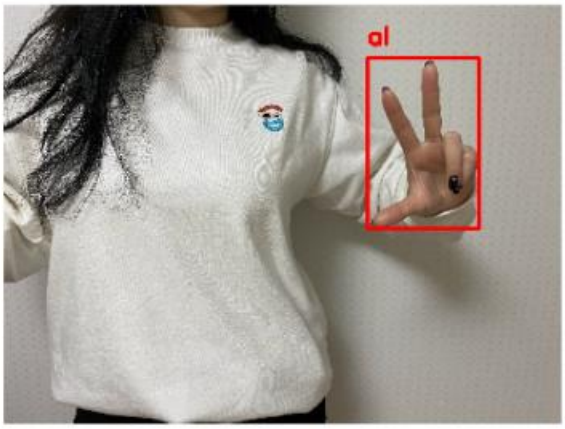
DataSet

The dataset used in the models consits of two files train and valid each file contains another two files image and labels. The dataset was preprocessed to handle missing values, normalize numerical features, and encode categorical variables.

```python
# Define transformations
train_transform = transforms.Compose([
    transforms.ToPILImage(),
    transforms.Resize((224, 224)),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
])

valid_transform = transforms.Compose([
    transforms.ToPILImage(),
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
])
```

And visualized as this





Evaluation Metrics

The following metrics were used to evaluate the models:

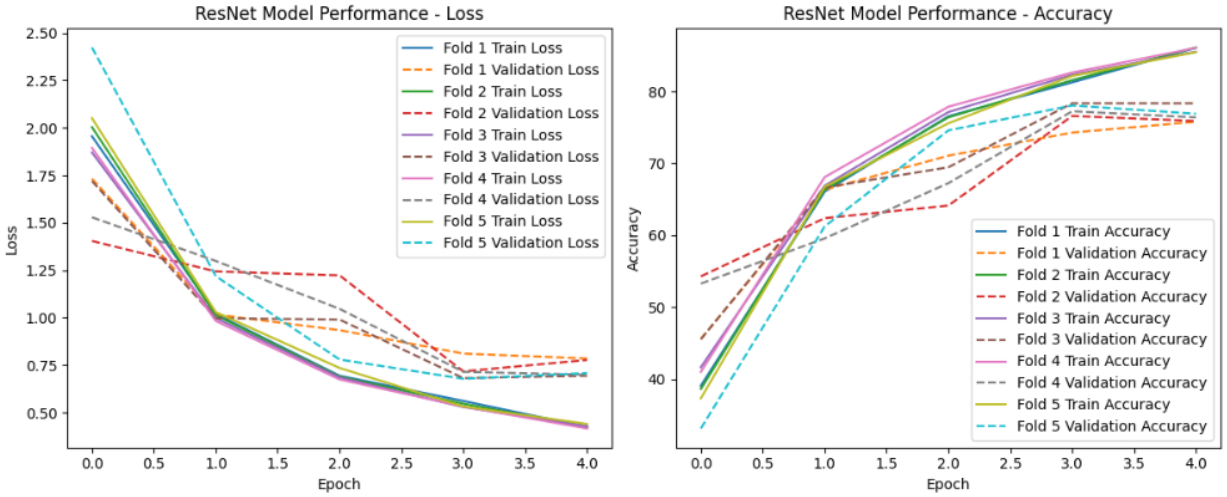
- Accuracy: Measures the overall correctness of predictions.
- Precision: Evaluates the proportion of true positive predictions among all positive predictions.
- Recall: Measures the model's ability to identify all relevant instances.

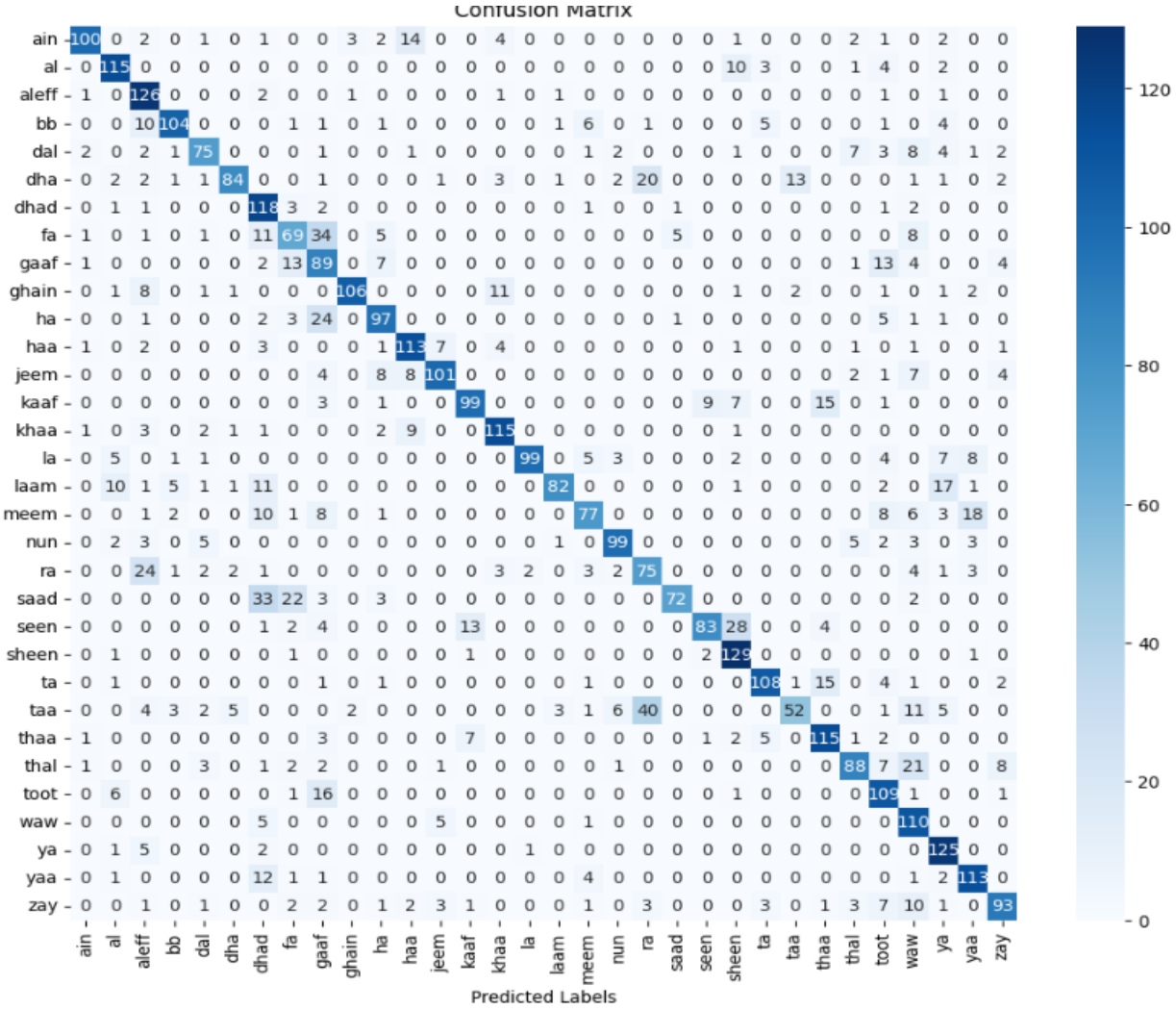
- F1-Score: Harmonic mean of precision and recall.

```python
# Evaluate on test data
model.eval()
all_preds = []
all_labels = []

with torch.no_grad():
    for images, labels in test_loader:
        images, labels = images.to(device), labels.to(device)
        outputs = model(images)
        _, predicted = torch.max(outputs, 1)
        all_preds.extend(predicted.cpu().numpy())
        all_labels.extend(labels.cpu().numpy())

# Compute confusion matrix
cm = confusion_matrix(all_labels, all_preds)

# Plot confusion matrix
plt.figure(figsize=(12, 10))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=class_names, yticklabels=class_names)
plt.title("Confusion Matrix")
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.show()

# Print classification report
print("\nClassification Report:")
report = classification_report(all_labels, all_preds, target_names=class_names, output_dict=True)
print(classification_report(all_labels, all_preds, target_names=class_names))

# Visualize precision, recall, and F1-score
metrics_df = pd.DataFrame(report).transpose()
metrics_df = metrics_df[['precision', 'recall', 'f1-score']].iloc[:-1]  # Exclude 'accuracy' row
metrics_df.plot(kind='bar', figsize=(16, 8), colormap='viridis')
plt.title("Classification Metrics (Precision, Recall, F1-Score)")
plt.ylabel("Score")
plt.xlabel("Class")
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()

# Print overall accuracy
test_accuracy = accuracy_score(all_labels, all_preds)
print(f"Test Accuracy: {test_accuracy:.2f}")
```

```python
# Function to plot performance (train vs validation accuracy and loss) for each fold
def plot_performance_by_fold(fold_histories, title):
    plt.figure(figsize=(12, 5))

    # Plot Loss for each fold
    plt.subplot(1, 2, 1)
    for fold_idx, fold_history in enumerate(fold_histories):
        plt.plot(fold_history['train_loss'], label=f'Fold {fold_idx + 1} Train Loss')
        plt.plot(fold_history['val_loss'], label=f'Fold {fold_idx + 1} Validation Loss', linestyle='dashed')
    plt.title(f"{title} - Loss")
    plt.xlabel("Epoch")
    plt.ylabel("Loss")
    plt.legend()

    # Plot Accuracy for each fold
    plt.subplot(1, 2, 2)
    for fold_idx, fold_history in enumerate(fold_histories):
        plt.plot(fold_history['train_acc'], label=f'Fold {fold_idx + 1} Train Accuracy')
        plt.plot(fold_history['val_acc'], label=f'Fold {fold_idx + 1} Validation Accuracy', linestyle='dashed')
    plt.title(f"{title} - Accuracy")
    plt.xlabel("Epoch")
    plt.ylabel("Accuracy")
    plt.legend()

    plt.tight_layout()
    plt.show()

# Call the function to plot metrics for each fold
plot_performance_by_fold(fold_histories, "ResNet Model Performance")

print("\nCross-validation completed.")
```

Resnet-18 Model Results


ResNet Model Performance - Loss
Loss
Epoch
Fold 1 Train Loss
Fold 1 Validation Loss
Fold 2 Train Loss
Fold 2 Validation Loss
Fold 3 Train Loss
Fold 3 Validation Loss
Fold 4 Train Loss
Fold 4 Validation Loss
Fold 5 Train Loss
Fold 5 Validation Loss
ResNet Model Performance - Accuracy
Accuracy
Epoch
Fold 1 Train Accuracy
Fold 1 Validation Accuracy
Fold 2 Train Accuracy
Fold 2 Validation Accuracy
Fold 3 Train Accuracy
Fold 3 Validation Accuracy
Fold 4 Train Accuracy
Fold 4 Validation Accuracy
Fold 5 Train Accuracy
Fold 5 Validation Accuracy

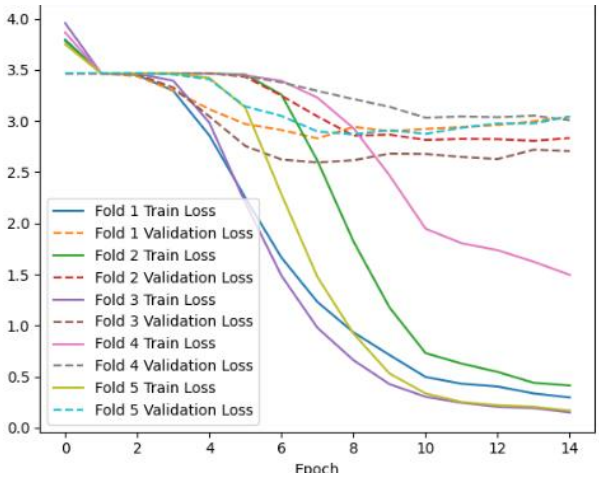

Confusion Matrix
ain 100 0 2 0 1 0 1 0 0 3 2 14 0 0 4 0 0 0 0 0 0 0 1 0 0 0 2 1 0 2 0 0
al 0 115 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 10 3 0 0 1 4 0 2 0 0
aleff 1 0 126 0 0 0 2 0 0 1 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0
bb 0 0 10 104 0 0 0 1 1 0 1 0 0 0 0 0 1 6 0 1 0 0 0 5 0 0 0 1 0 4 0 0
dal 2 0 2 1 75 0 0 0 1 0 0 1 0 0 0 0 0 1 2 0 0 0 1 0 0 0 7 3 8 4 1 2
dha 0 2 2 1 1 84 0 0 1 0 0 0 1 0 3 0 1 0 2 20 0 0 0 0 13 0 0 0 1 1 0 2
dhad 0 1 1 0 0 0 118 3 2 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 1 2 0 0 0
fa 1 0 1 0 1 0 11 69 34 0 5 0 0 0 0 0 0 0 0 0 5 0 0 0 0 0 0 0 8 0 0 0
gaaf 1 0 0 0 0 0 2 13 89 0 7 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 13 4 0 0 4
ghain 0 1 8 0 1 1 0 0 0 106 0 0 0 0 11 0 0 0 0 0 0 0 1 0 2 0 0 1 0 1 2 0
ha 0 0 1 0 0 0 2 3 24 0 97 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 5 1 1 0 0
haa 1 0 2 0 0 0 3 0 0 0 1 113 7 0 4 0 0 0 0 0 0 0 1 0 0 0 1 0 1 0 0 1
jeem 0 0 0 0 0 0 0 0 4 0 8 8 101 0 0 0 0 0 0 0 0 0 0 0 0 0 2 1 7 0 0 4
kaaf 0 0 0 0 0 0 0 0 3 0 1 0 0 99 0 0 0 0 0 0 0 9 7 0 0 15 0 1 0 0 0 0
khaa 1 0 3 0 2 1 1 0 0 0 2 9 0 0 115 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
la 0 5 0 1 1 0 0 0 0 0 0 0 0 0 0 99 0 5 3 0 0 0 2 0 0 0 0 4 0 7 8 0
laam 0 10 1 5 1 1 11 0 0 0 0 0 0 0 0 0 82 0 0 0 0 0 1 0 0 0 0 2 0 17 1 0
meem 0 0 1 2 0 0 10 1 8 0 1 0 0 0 0 0 0 77 0 0 0 0 0 0 0 0 0 8 6 3 18 0
nun 0 2 3 0 5 0 0 0 0 0 0 0 0 0 0 0 1 0 99 0 0 0 0 0 0 0 5 2 3 0 3 0
ra 0 0 24 1 2 2 1 0 0 0 0 0 0 0 3 2 0 3 2 75 0 0 0 0 0 0 0 0 4 1 3 0
saad 0 0 0 0 0 0 33 22 3 0 3 0 0 0 0 0 0 0 0 0 72 0 0 0 0 0 0 0 2 0 0 0
seen 0 0 0 0 0 0 1 2 4 0 0 0 0 13 0 0 0 0 0 0 0 83 28 0 0 4 0 0 0 0 0 0
sheen 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 2 129 0 0 0 0 0 0 0 1 0
ta 0 1 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 1 0 0 0 0 0 108 1 15 0 4 1 0 0 2
taa 0 0 4 3 2 5 0 0 0 2 0 0 0 0 0 0 3 1 6 40 0 0 0 0 52 0 0 1 11 5 0 0
thaa 1 0 0 0 0 0 0 0 3 0 0 0 0 7 0 0 0 0 0 0 0 1 2 5 0 115 1 2 0 0 0 0
thal 1 0 0 0 3 0 1 2 2 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 88 7 21 0 0 8
toot 0 6 0 0 0 0 0 1 16 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 109 1 0 0 1
waw 0 0 0 0 0 0 5 0 0 0 0 0 5 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 110 0 0 0
ya 0 1 5 0 0 0 2 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 125 0 0
yaa 0 1 0 0 0 0 12 1 1 0 0 0 0 0 0 0 0 4 0 0 0 0 0 0 0 0 0 0 1 2 113 0
zay 0 0 1 0 1 0 0 2 2 0 1 2 3 1 0 0 0 1 0 3 0 0 0 3 0 1 3 7 10 1 0 93
ain al aleff bb dal dha dhad fa gaaf ghain ha haa jeem kaaf khaa la laam meem nun ra saad seen sheen ta taa thaa thal toot waw ya yaa zay
Predicted Labels
120
100
80
60
40
20
0

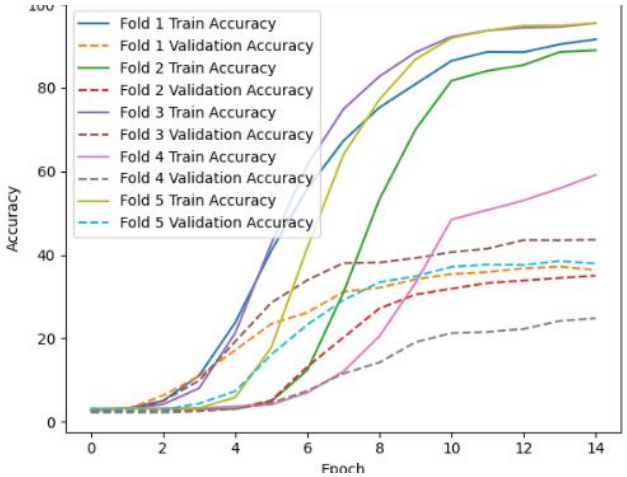
Classification Report:

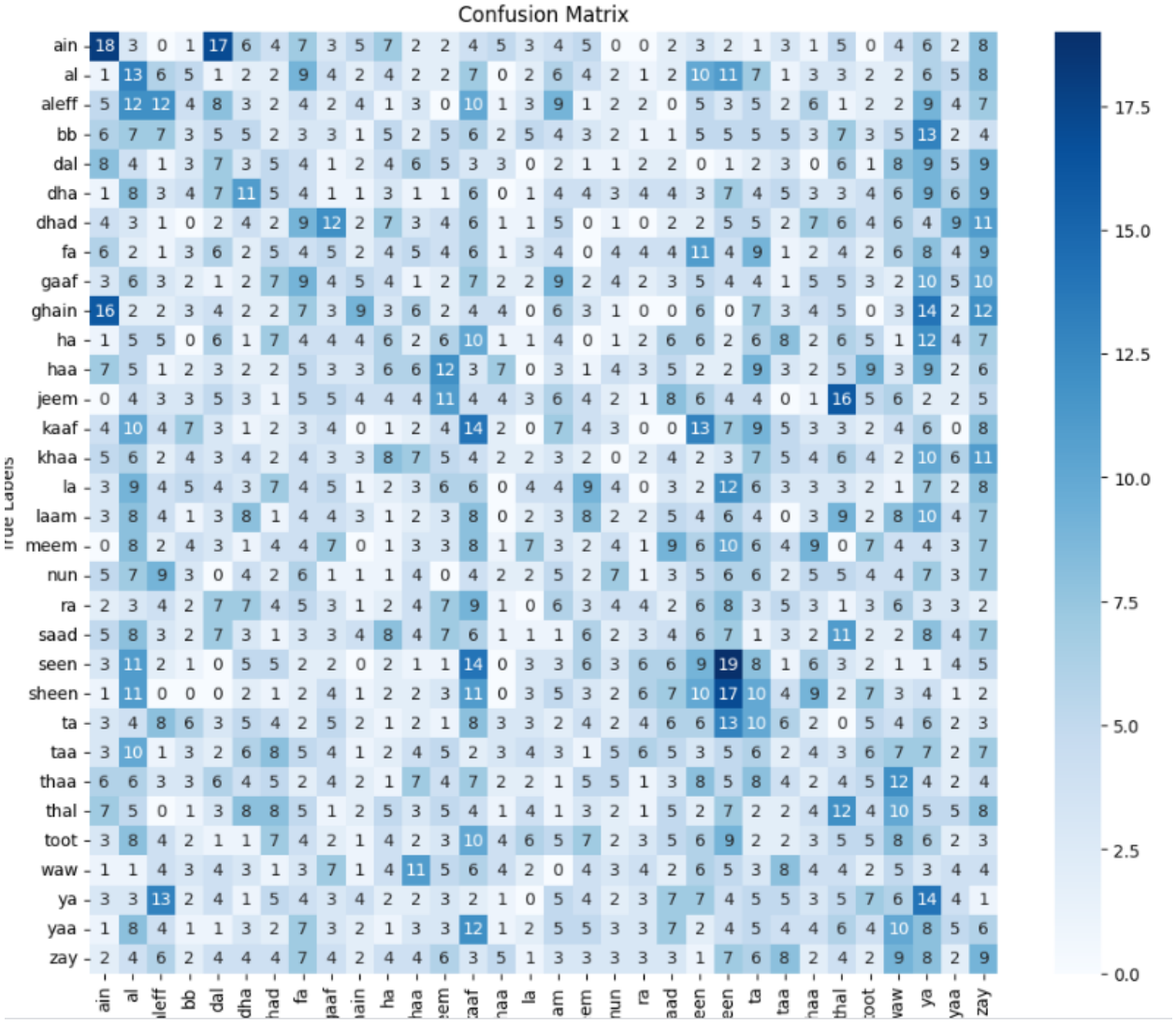
	precision	recall	f1-score	support
ain	0.92	0.75	0.83	133
al	0.79	0.85	0.82	135
aleff	0.64	0.94	0.76	134
bb	0.88	0.77	0.82	135
dal	0.78	0.68	0.72	111
dha	0.89	0.62	0.73	135
dhad	0.55	0.91	0.68	130
fa	0.57	0.51	0.54	135
gaaf	0.45	0.66	0.53	134
ghain	0.95	0.79	0.86	135
ha	0.75	0.72	0.73	135
haa	0.77	0.84	0.80	135
jeem	0.86	0.75	0.80	135
kaaf	0.82	0.73	0.77	135
khaa	0.82	0.85	0.83	135
la	0.97	0.73	0.84	135
laam	0.92	0.62	0.74	132
meem	0.76	0.57	0.65	135
nun	0.86	0.80	0.83	123
ra	0.54	0.61	0.57	123
saad	0.91	0.53	0.67	135
seen	0.87	0.61	0.72	135
sheen	0.70	0.96	0.81	135
ta	0.87	0.80	0.83	135
taa	0.76	0.39	0.51	135
thaa	0.77	0.84	0.80	137
thal	0.79	0.65	0.72	135
toot	0.61	0.81	0.70	135
waw	0.54	0.91	0.68	121
ya	0.71	0.93	0.80	134
yaa	0.75	0.84	0.79	135
zay	0.79	0.69	0.74	135
accuracy			0.74	4247
macro avg	0.77	0.74	0.74	4247
weighted avg	0.77	0.74	0.74	4247

Overall Accuracy is 0.82

HandCraftedCNN Model Results


Fold 1 Train Loss
Fold 1 Validation Loss
Fold 2 Train Loss
Fold 2 Validation Loss
Fold 3 Train Loss
Fold 3 Validation Loss
Fold 4 Train Loss
Fold 4 Validation Loss
Fold 5 Train Loss
Fold 5 Validation Loss
Epoch


Fold 1 Train Accuracy
Fold 1 Validation Accuracy
Fold 2 Train Accuracy
Fold 2 Validation Accuracy
Fold 3 Train Accuracy
Fold 3 Validation Accuracy
Fold 4 Train Accuracy
Fold 4 Validation Accuracy
Fold 5 Train Accuracy
Fold 5 Validation Accuracy
Accuracy
Epoch


Confusion Matrix
True Labels
ain
al
aleff
bb
dal
dha
dhad
fa
gaaf
ghain
ha
haa
jeem
kaaf
khaa
la
laam
meem
nun
ra
saad
seen
sheen
ta
taa
thaa
thal
toot
waw
ya
yaa
zay

Classification Report:

	precision	recall	f1-score	support
ain	0.13	0.14	0.13	133
al	0.06	0.10	0.08	135
aleff	0.10	0.09	0.09	134
bb	0.04	0.02	0.03	135
dal	0.05	0.06	0.06	111
dha	0.09	0.08	0.09	135
dhad	0.02	0.02	0.02	130
fa	0.03	0.03	0.03	135
gaaf	0.03	0.03	0.03	134
ghain	0.12	0.07	0.09	135
ha	0.05	0.04	0.05	135
haa	0.05	0.04	0.05	135
jeem	0.08	0.08	0.08	135
kaaf	0.07	0.10	0.08	135
khaa	0.03	0.01	0.02	135
la	0.06	0.03	0.04	135
laam	0.02	0.02	0.02	132
meem	0.02	0.01	0.02	135
nun	0.08	0.06	0.07	123
ra	0.05	0.03	0.04	123
saad	0.03	0.03	0.03	135
seen	0.05	0.07	0.06	135
sheen	0.08	0.13	0.10	135
ta	0.06	0.07	0.06	135
taa	0.02	0.01	0.02	135
thaa	0.02	0.01	0.02	137
thal	0.08	0.09	0.08	135
toot	0.04	0.04	0.04	135
waw	0.03	0.04	0.04	121
ya	0.06	0.10	0.08	134
yaa	0.05	0.04	0.04	135
zay	0.04	0.07	0.05	135
accuracy			0.06	4247
macro avg	0.05	0.06	0.05	4247
weighted avg	0.05	0.06	0.05	4247

Overall Accuracy is 6%

Summary.

The resnet model works just fine and there is no overfitting while the CNN model graphs show an obvious overfitting since the training accuracy is very high while the validation accuracy is very low